TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MIKA LATVA-KYYNY

DEVELOPING A RELATIONAL DATABASE APPLICATION PROTOTYPE FOR DETAILED INSTRUMENTATION ENGINEERING

Master of Science Thesis

Examiner: Professor Matti Vilkko

Examiner and topic approved by the Faculty Council of the Faculty of Engineering Sciences on 13th of January 2016

ABSTRACT

MIKA LATVA-KYYNY: Developing a relational database application prototype for detailed instrumentation engineering
Tampere University of Technology
Master of Science Thesis, 63 pages, 52 Appendix pages
June 2016
Master's Degree Programme in Automation Technology
Major: Process Automation Technology
Examiner: Professor Matti Vilkko

Keywords: instrumentation engineering, relational database, Unified Modeling Language, Microsoft Access

The research problem of this thesis was to study how to dispose of the disadvantages of the file-system approach in information control of detailed instrumentation engineering data. The main objectives of the solution were to reduce data redundancy and separate data from report templates. The problem was solved for one case project's engineering data by developing a relational database application prototype designed to manage detailed instrumentation data of the case project. The objective was also to find and document a systematic development process and data models that can be used in future projects for developing a new project-specific database application for information control.

The development process started from defining the initial requirements for the database application by analyzing case project's reports. UML was used to develop the use cases and conceptual schema. The application was implemented by using Microsoft Access and tested by using ad-hoc and model-based testing.

The database application prototype developed in this thesis was able to hold all data of the case project with minimal redundancy and separation between data and report templates. Compared to the file-based approach of the case project, it could be possible to save time, reduce likelihood for errors, and allow multiple users accessing the data simultaneously by using the developed database application. The documented design process and data models of this thesis can be used to develop new applications for future projects if schedules of the projects enable careful database design to be done.

TIIVISTELMÄ

MIKA LATVA-KYYNY: Relaatiotietokantapohjaisen sovellusprototyypin kehittäminen instrumentoinnin detaljisuunnitteluun
Tampereen teknillinen yliopisto
Diplomityö, 63 sivua, 52 liitesivua
Kesäkuu 2016
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: prosessiautomaatio
Tarkastaja: professori Matti Vilkko

Avainsanat: instrumentointisuunnittelu, relaatiotietokanta, UML, Microsoft Access

Tämän diplomityön tutkimusongelmana oli selvittää, miten päästään eroon tiedostoihin perustuvien järjestelmien epäedullisuuksista instrumentoinnin detaljisuunnittelun informaationhallinnassa. Tutkimusongelman ratkaisun päätavoitteena oli vähentää redundantin datan määrää sekä erottaa data raporttipohjista. Tutkimusongelma ratkaistiin yhden case-projektin suunnitteludatalle kehittämällä relaatiotietokantapohjainen sovellusprototyyppi, jolla kyettiin hallitsemaan case-projektin instrumentointisuunnittelun tuottamaa dataa. Työn tavoitteena oli myös löytää ja dokumentoida systemaattinen suunnitteluprosessi sekä käsitteellinen malli siten, että tätä tietoa voitaisiin hyödyntää kehitettäessä tulevaisuudessa tietokantapohjaisia sovelluksia uusien projektien informaatiohallinnan tarpeisiin.

Suunnitteluprosessi aloitettiin määrittelemällä ensivaatimukset tietokantasovellukselle analysoimalla case-projektin suunnitteluraportteja. Käyttötilannemallien ja käsitteellisen mallin luomiseen käytettiin UML-mallinnuskieltä. Sovellus toteutettiin käyttäen Microsoft Access -sovellusta ja testattiin käyttäen ad-hoc-testausta sekä mallipohjaista testausmenetelmää.

Tässä diplomityössä kehitetty tietokantasovellus kykeni hallitsemaan kaikkea case-projektin dataa minimaalisella redundantin datan määrällä sekä pitämään datan erillään raporttipohjista. Case-projektin tiedostoihin perustuviin järjestelmiin pohjautuvaan lähestymistapaan verrattuna tässä diplomityössä kehitetyllä tietokantasovelluksella olisi mahdollista säästää aikaa, pienentää virheiden todennäköisyyttä sekä mahdollistaa se, että useat eri käyttäjät pääsevät yhtä aikaa käsittelemään samaa dataa. Tähän työhön dokumentoitua suunnitteluprosessia sekä käsitteellistä mallia voidaan käyttää vastaavien sovellusten kehittämiseen uusissa projekteissa, jos projektien aikatauluissa on varaa huolelliselle tietokantasuunnittelulle.

TABLE OF CONTENTS

1. INTRODUCTION 1
   1.1 Background 1
   1.2 Advantages of a Database Approach 2
   1.3 Scope and Limitations 4
   1.4 Outline of the Thesis 4
2. RELATIONAL DATABASE THEORY 5
   2.1 Database Terminology, Concepts and Architecture 5
   2.2 Conceptual Database Modeling with UML 7
   2.3 The Relational Data Model and Relational Database Constraints 12
   2.4 SQL 15
   2.5 Database Design Process 17
   2.6 Microsoft Access 28
3. DESIGN AND IMPLEMENTATION 31
   3.1 Requirements Definition 31
   3.2 Conceptual Database Design 49
   3.3 Logical Database Design 49
   3.4 Implementation 51
4. VALIDATION AND TESTING 55
   4.1 Ad-Hoc and Model-Based Testing 55
   4.2 Validation of the Solution for the Research Problem 56
5. CONCLUSION 60

APPENDIX A: Sample Use Case Template
APPENDIX B: Use Case Diagrams for the Database Application Prototype
APPENDIX C: Use Case Descriptions for the Database Application Prototype
APPENDIX D: Class Diagrams for the Database Application Prototype
APPENDIX E: Relational Model for the Database Application Prototype
APPENDIX F: Test Case Specification for the Database Application Prototype

LIST OF ACRONYMS

ADO	ActiveX data object
ATM	Automated Teller Machine
DAO	Data Access Object
DBA	Database Administrator
DBMS	Database Management System
DCS	Distributed Control System
DDL	Data Definition Language
DML	Data Manipulation Language
DOL	Direct On Line
IDE	Instrumentation Design Engineer
IIBA	International Institute of Business Analysis
ISBL	Inside Battery Limits
ISO	International Standards Organization
I/O	Inputs and Outputs of a control system
MCC	Motor Control Center
OMG	Object Management Group
OSBL	Outside Battery Limits
PFD	Process Flow Diagram
PLC	Programmable Logic Controller
P&ID	Process and Instrumentation Diagram
SEQUEL	Structured English Query Language
SQL	Structured Query Language
RDBMS	Relational Database Management System
RVBA	Reddick VBA
SDL	Storage Definition Language
SIL	Safety Integrity Level
SIS	Safety Instrumented System
SS	Soft Starter
UML	Unified Modeling Language
VDL	View Definition Language
VFD	Variable Frequency Drive
VBA	Visual Basic for Applications
2-S	2-speed Motor

LIST OF SYMBOLS

$A_{i,}$ a_i	i:th attribute of a relation
C	Superclass of generalization
$dom(A_i)$	Domain of attribute A_i
FK	Foreign key
h	Hour
L	Relation
n	Number of attributes in a relation
PK	Primary key
r	Same as $r(R)$
$r(R)$	Relation state
R	Represents name of a relation
R_1, R_2	Relation schemas
S_i	Sub class of generalization
t_i	i:th tuple of a relation
$t[A_i]$	Data value of attribute A_i in tuple t
v_i	i:th attribute of a tuple being an element of $dom(A_i)$ or null
*	In SQL queries asterisk is a wildcard character. In UML, asterisk represents any number of objects for cardinality or optionality constraints of associations between class diagrams.
°C	Celsius Degree

1. INTRODUCTION

This Master's thesis describes the development of a relational database application prototype to support process instrumentation in detailed engineering projects. The thesis comprises designing the database structure by using the data of a real-world case project. The development process includes using systematic database design methods, and implementing the design with Access™, a database application from Microsoft. The objective of this thesis is to design a prototype of the database structure and user interface so that this information can be used to create a database application for some projects that do not have any other database application available. This chapter first introduces the background of the thesis, then describes the motivation and challenges of the work, then explains the scope and limitations of the topic, and finally introduces the outline by shortly describing each chapter.

1.1 Background

Big detailed engineering projects create a great amount of design data that engineers need to store somewhere. Using traditional spreadsheet applications or other file based applications to store the data results in much inefficiency, such as data redundancy. In big projects, a database application is more likely to be a solution. Properly designed relational database is the most efficient way to store and retrieve the design data in instrumentation engineering projects [16, pp. 18]. However, there is not always a ready database application available. The research problem of this thesis is to study how to dispose of the disadvantages of the file-system approach by designing a relational database application for the case project. Two main disadvantages of the case project's file-system approach were that there was big amount of redundant data and that data was too strictly bound to documents making it difficult to change document layouts. Thus, the main target of this thesis is to minimize the amount of redundant data and separate data from document templates.

There are many commercially available database applications available for detailed engineering projects but they can be big investments for an engineering company. There is also a risk that commercial applications are not flexible enough to control different data in various projects. In addition, the company purchasing a commercial application becomes dependent on the company who developed it because when the user needs of the application change, the application requires updates and redevelopment. Another option for the engineering company is to develop an own database application. For example, Lipták introduces some examples of a database approach for instrumenta-

tion and control engineering in his book Instrumentation Engineers' Handbook [16]. Some of his examples are used in this thesis. The benefits for developing an own application is that, in case of a simple application, it is likely to be inexpensive to develop and maintain. The challenge in developing such application is that even though instrumentation-engineering data is relatively similar in all projects, each client has some variations in their data. For this reason, it is challenging to develop a single application that would be suitable for all projects. This thesis addresses the research problem by providing a solution to develop a database application for one case project by using systematic database design methods. Then, when there is a need for a database application in some other project, the company can use the documented information to re-develop the application with modifications to respond the project's needs. When there is ready example available, the time needed to design the database structure and user interface can remarkably decrease compared to starting from scratch.

In this thesis, the case project was a big factory expansion project in which the author was involved as an employee of the engineering company providing engineering, procurement, and construction management services to project's client. The scope of the instrumentation engineering in this project comprehended over eight hundred new field instruments and almost hundred new electric motors. These devices had altogether over thousand five hundred inputs and outputs (I/O) to four different control systems: two distributed control systems (DCS) and two programmable logic controllers (PLC). Other engineering disciplines involved to the project were process engineering, civil engineering, mechanical engineering, piping and layout engineering, automation engineering, and electrical engineering.

Defining and implementing a database requires a database management system (DBMS). One definition for the DBMS is that "the DBMS is a general-purpose software system that facilitates the process of defining, constructing, manipulating, and sharing databases among various users and applications". [1, pp. 5] There are two main types of DBMS: object DBMS and relational DBMS. [3, pp. 500] The latter was chosen for this thesis because Access was used for implementing the database application, and Access is a relational database management system (RDBMS). The reason to implement the database application with Access was that in the engineering company, all employees had it readily available on their laptop.

1.2 Advantages of a Database Approach

This chapter introduces the advantages of a database approach versus a file-system approach. Using flat files for storing data can result in a number of problems. Each file holds data for a specific purpose, but some of the data may be redundant within a single file as well as between individual files. [3, pp. 500] This redundant data results in unnecessary storage space and redundant efforts to keep the data synchronized between the files. [1, pp. 9–10] In addition, if someone wants to change the layout of a docu-

ment, or make a new document containing data from different files, they cannot implement these changes without a considerable amount of work. [3, pp. 511] The motivation of this thesis is to provide a feasible alternative for projects, which engineers would otherwise carry out using file systems.

There are four main characteristics of the database approach versus the file-systems approach. First, database systems are self-describing by their nature. This means that the database system contains not just the database itself but also a complete structure and constrains of the database. In comparison, file systems have their data structure and constrains typically defined individually within each application. Second characteristic is that database systems provide data abstraction, which enables users to make changes to data structure without anyhow affecting the application programs accessing the data. This is not usually the case with file systems. Third characteristic is that databases allow data sharing and multiuser transaction processing. Transaction is an executing program or process which includes one or more database accesses, such as reading or updating of database records. Fourth characteristic of database approach is that databases allow multiple views of data. There can be many users, each viewing a different subset of data simultaneously. [1, pp. 10–14] Even though some file-system applications, such as Microsoft Excel, allow file sharing and multiple views between users, it is not even nearly as flexible and secure to do as with databases.

Database applications make it easier to store, retrieve, and maintain data because with databases it is easier to find, search, sort, arrange, and link data in more versatile way than with traditional file-based documentation. In order to get full benefit out of the database approach, it should be used from the very beginning of the project. [16, pp. 22] When all data is stored only once and in a single place, the time needed to update the data decreases remarkably. The greater the number of places where the same information is needed, the greater the benefit. A likelihood for an error or unsynchronized data decreases. Team members need less communication with each other because they are all sharing the same data. This can decrease unnecessary distractions and email correspondence between the team members. This effect is likely to be even greater if the database is common for multiple engineering disciplines. Moreover, changing a layout or data content of, for example, eight hundred instrument datasheets requires nothing more than simply making the required changes to a datasheet template and generating a new set of datasheets. However, there are some challenges when choosing the database approach.

Unlike with spreadsheet applications, with database applications, the user cannot just start from typing in the data. Database applications require that the database structure and user interface is first carefully designed and implemented before the data can be added. Good database design can take time [17] but projects tend to have tight schedules. In engineering projects, the type and structure of data varies between clients and sometimes even between projects of the same client. It would be challenging to try

to develop a database that is suitable for the data of many different clients and projects. There are many commercial database applications available for engineering purposes but these are relatively expensive and there is a risk that they are not flexible enough to respond to all needs of the project. For this reason, the objective in this thesis is to develop a database that is project specific.

1.3 Scope and Limitations

The scope of this thesis is to develop a database application for producing the following four instrumentation document types that an engineering company usually delivers to a client: instrument lists, I/O lists, cable lists, and instrument datasheets. In addition to these instrumentation documents, the application should be also capable of producing process equipment list, pipeline list, and motor list. These document types belong to process and electrical engineering disciplines but their data is very much related to instrumentation engineering documents. Producing any other types of deliverable documents than those mentioned above are beyond the scope of this thesis. The scope of the application prototype does not also include user authentication or automatic revision control even though they could be useful features in this kind of application.

Software can be defined to include three primary components: *instructions* forming computer program, *data structure* defining how the information is stored for the manipulation and transformation of the computer program, and *documentation* describing the operation and use of the software. [10, pp. 356] The scope of this thesis includes these three components excluding the user manual due to the reason that the objective is to develop only an application prototype.

1.4 Outline of the Thesis

Chapter 2 introduces relational database theory, database design process, and basics of Microsoft Access which are the foundation of this thesis. The relational database theory comprehends the mathematical foundation of the relational databases, and database programming language, structured query language (SQL). The database design process starts from requirements definition and analysis and continues with functional requirements, conceptual design, logical design, implementation, and finally validation and testing. Basics of Microsoft Access introduce the four main Access objects: tables, queries, forms, and reports, and also Access programming with macros and VBA. Chapter 3 goes through the design of the database application in practice following the steps of the database design process from requirements definitions to implementation. In chapter 4, the application is tested and the solution to the research problem is evaluated. Finally, chapter 5 concludes the results of this thesis.

2. RELATIONAL DATABASE THEORY

Before we can start discussing about databases, we need to be familiar with the database terminology and basic architecture of databases. Then we continue with conceptual data modeling, which helps us to understand database structures. Class diagrams of the Unified Modeling Language are introduced as a tool that we can use for these modeling purposes. The modeling provides us with data abstraction – the fundamental characteristic of the database approach. When you are familiar with conceptual data modeling, it is time to introduce how conceptual model is mapped into relational data model. Next, structured query language used in database management systems is also introduced. Database design process is then walked-through and, finally, the basics of Microsoft Access are shortly introduced.

2.1 Database Terminology, Concepts and Architecture

What do we mean with a database? One definition is that a database is a collection of related data, and it has the following three characteristics: it represents some aspect of a real world (also called as a *miniworld*), its data is logically coherent and has some inherent meaning, and it has a group of users using it for some specific purpose with DBMS. [1, pp. 4] [4] The DBMS is typically divided into two modules: client module and server module. Client module (also called a front-end) runs on user's computer and contains the user interface for the application. Server module (also called a back-end) works as a data storage and handles among others search and access. [1, pp. 29] In the following sections, we discuss about the basic database terminology, architecture, and languages.

2.1.1 Basics of Database Terminology

When we want to use a high-level data model called conceptual model, we discuss in terms such as *entities*, *attributes* and *relationships*. When the miniworld is the aspect of the real-world that we want to describe in our database, entities are the objects or concepts of this miniworld. [1, pp. 31] If we wanted to make a database to manage, for example, the data for a football tournament, we would have entities such as players and teams. Attributes represent the properties of these entities. [1, pp. 31] Football players can have attributes such as name, age and address. A group of similar entities sharing the same attributes are called *an entity type*. [1, pp. 65] Relationships represent the associations between the entities. For example, players and teams have relationships between each other because each player *plays in* a team. A set of relationships between different entity types are called *a relationship type*. [1, pp. 70]

It is important to distinguish between description of a database and database itself. The description of the database describes the structure of the data and it is called the *database schema*. When we design a database, we define the database schema. Once it is designed and implemented in the DBMS, it is not expected to change very frequently. It only changes when the requirements for our database application change. The actual data stored in the database, is on the other hand changing every time something changes in the miniworld and we need to reflect these changes to our database. The current data in the database at any given time is called the *database state*. When we make changes to the data, the database state is changing. [1, pp. 32]

2.1.2 Three-Schema Architecture

The three-schema architecture is a way to achieve the following three important characteristics of the database: insulation between application programs and data, support of multiuser views, and the possibility of storing the database schema into DBMS. Figure 1 represents a model of this three-schema architecture. [1, pp. 33]

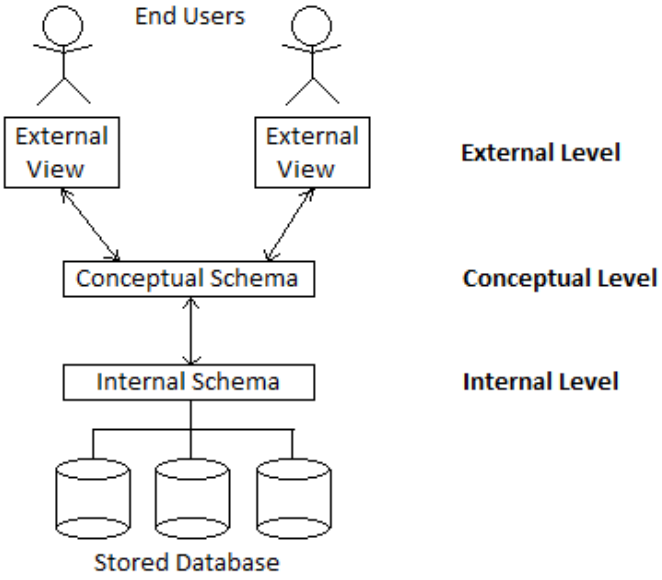


Figure 1. *The three-schema architecture of a database. [1, pp. 34]*

At the bottom level of the three-schema architecture, there is the internal level. This level contains the internal schema, which describes how the data is physically stored in the database. Above the internal level, there is the conceptual schema that defines the data structure of the whole database. This level is distinct from the internal level and describes the database only in terms of entities, attributes, relationships, and the constraints they have. In database design, a high-level conceptual model is usually used to implement this level of the database. At the top of this database architecture, there is the external level, which contains the external view (or external schema) of the database. At this level, there are the application programs that the database users use to interact with

the database. Each user group can have their own view of the database, which they use to interact only with that subset of the data that interests them. This level of the database can be implemented by using a high-level external model of the database. The three-schema architecture is a useful way to visualize the database concept based on different schema levels. Even though most DBMS software do not completely separate these three levels from each other, they are still usually partially based on this architecture. [1, pp. 34–35]

2.1.3 Database Languages

In true three-schema architecture, we would need three different languages to implement our database: storage definition language (SDL) would define the internal schema, data definition language (DDL) would define the conceptual schema, and view definition language (VDL) would define the external schema. If there is not very strict separation between the different schema levels, there is no need for the SDL but the DDL is used to define both the internal and the conceptual schema. Even if there is a strict separation between the different schema levels, the specific SDL is not very common. In current relational DBMS products, the internal level is defined in the means of parameters and specifications.

Then when the database is implemented and populated with data, the fourth language called data manipulation language (DML) is needed to update, retrieve, delete and insert the data. In practice, the DDL, VDL, and DML are not three distinct languages in current DBMS products. Rather, there is a comprehensive language integrating all these three languages. The most common such language for the relational databases is Structured Query Language (SQL). We discuss the SQL in more detail in Section 2.4.

2.2 Conceptual Database Modeling with UML

Conceptual database modeling is a very important phase in designing a database application. The purpose of the model is to define a conceptual database schema that is usually independent of a specific DBMS implementation. This way the restrictions of a specific DBMS do not influence the design of a database structure. The created model is then invaluable description of the database, which can be used to implement the database with any DBMS. [1, pp. 57, 413] This section introduces the schema design by using class diagrams of Unified Modeling Language (UML). The diagrams provide us with a way to visualize and abstract features of the design. Abstraction simplifies complex system and allows us to focus only on the main features of the system. [7, pp. 12]

The UML is the standard language for modeling object-oriented systems in the field of software engineering. [7, pp. 5] The UML has a variety of different diagram types for different needs in a software development process. [1, pp. 84] It provides

software developers a way to communicate in standard language if they are working as a team. The UML standard is managed by the Object Management Group (OMG). The first versions of the standard were developed in 1990s by unifying the best features of a number of analysis and design techniques being in use that time. [7, pp. 14]

Even though the UML was developed for object-oriented system modeling, it is also relevant for relational database design. Class diagrams of the UML are in many ways similar to traditional entity-relationship diagrams that are used in database modeling. [1, pp. 58] In Section 2.5.2, we discuss also another type of UML diagram, use cases.

2.2.1 Class Diagrams

With UML class diagrams, a database entity type can be described as *a class*, which is displayed as a rectangle with three panels (also called *compartments*). In the top panel, there is the name of the class. Entities are called objects in UML terminology and they represent the real-world instances of the class that describes their schema. Figure 2 illustrates the class diagrams notation for an example data of a company.

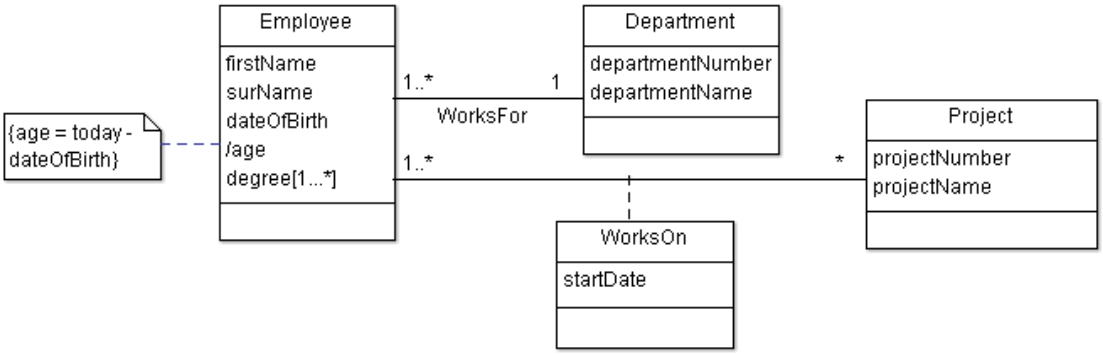


Figure 2. *UML class diagram notation for an example data of a company database. [1, pp. 84] [7, pp. 50]*

The name of a class can consist of more than one word which are written together without spaces and they should all start with a capital letter. The middle panel contains a list of all attributes that belong to the class. The attribute names should start with a lower-case letter and, if the name consists of multiple words, the words following previous words should start with a capital letter. The bottom panel of a class is used for *methods*, which are processes that a class is responsible for carrying out, but they are not in particular interest in this thesis. Showing the class name is compulsory but showing the list of attributes or methods is optional. If there are no attributes or methods to show, the particular panel can be left empty or excluded completely from the class. [1, pp. 84–85] [8, pp. 15, 18] [7, pp. 45,48]

2.2.2 Associations, Constraints and Link Attributes

In UML, relationship types between classes are called *associations* and relationships are called *links*. The associations are represented as lines between classes and they may optionally have a name. The associations have two important constraints that are derived from the miniworld. The first constraint is called *the cardinality* and it describes the maximum number of objects of a class that can be associated with the object of the other class. The second constraint is called *the optionality* and it describes the minimum number of objects of a class that can be associated with the object of the other class. These constraints are represented as *min…max* in the both ends of the association (min represents the optionality and max represents the cardinality). Asterisk (*) is used to indicate any number of objects in a constraint. The following ways to truncate constraints are used:

- 0…* is truncated to * and
- 1…1 is truncated to 1.

If the cardinality of an association is * in both ends of the association of two classes, the classes are said to have *a many-to-many relationship*. If the cardinality is 1 in one end and * in the other end, the classes have *a one-to-many relationship*. Again, if the cardinality is 1 in both ends, the classes have *a one-to-one relationship*. [1, pp. 70–85] [8, pp. 18]

The associations are read in both directions. Being A and B two classes. If we want to know how many objects of class B one particular object of class A is associated with, we need to look at the constrains on the B side. For example, Figure 2 defines that each employee is not necessarily associated with any project and at most, each employee can be associated with unlimited amount of projects. Each project on the other hand must have at least one employee and at most, each project can have any number of employees. The associations can also have attributes called *link attributes*. These link attributes are modeled by drawing a class containing these attributes and connecting this class to the related association line with a dashed line. In Figure 2, there is an association between classes *Employee* and *Project*. The name of the association is *WorksOn* and it has one link attribute called *startDate*. [1, pp. 70–85] [8: pp. 18]

2.2.3 Attribute Types

Different ways to classify attributes exist and we will now discuss two of them. First, an attribute that can have only a single value is called a *single-valued attribute*. In contrast, an attribute that can consist of multiple values is called *a multi-valued attribute*. In UML, multi-valued attributes are modeled by adding *a multiplicity clause* enclosed in square brackets immediately after the attribute name. The multiplicity clause consists of a lower and upper bound of the range representing the number of possible

values for the attribute. In Figure 2, class *Employee* has a multi-valued attribute *degree* representing all degrees that an employee can have. In this case, the range is [1…*] meaning that an employee must have at least one degree and there is no upper bound for the number of degrees an employee can have. If the range is [0…*], it is truncated to [*].

The second way to classify attributes is *derived* and *stored* attributes. If an attribute type is a derived attribute, it means we can determine the value of the attribute from the values of other attributes. Because we can derive this value when needed, it is not usually necessary that we store the attribute to the database. If the attribute is not derived, it called a stored attribute. We can model derived attributes by adding a forward slash (/) immediately before the attribute name. We can specify the derivation in a note attached to the derived attribute. Figure 2 illustrates the derived attribute *age* of an employee. In the note, the derivation is specified as a difference between the current date and the employee's date of birth. [1, pp. 61–64, 85] [7, pp. 50]

2.2.4 Generalization and Specialization

Sometimes, we can find that we have a class of which attributes do not apply to all objects of the class. For example, we might have a class representing information about persons in a university database. Some of the persons are employees and some are students. Then we notice that an attribute representing salary applies to employees but not for students and an attribute representing student records applies to students but not for employees. If we have these attributes in the same class, we need to leave them empty or write for example "not applicable" in case the attribute does not apply to the object. In order to achieve a reliable database, it is not good to have classes where it is possible to add inconsistent data to the objects (for example, adding salary for a student). [7, pp. 95–97]

One way to solve the problem mentioned in the previous paragraph is to use a method called *specialization*. In specialization, we create two *subclasses* for the class representing the persons in the university database. Both students and employees get an own subclass. The class representing the persons is then called a *superclass*. Figure 3 illustrates this specialization.

In specialization, the superclass has attributes that are common for all subclasses. In our example in Figure 3, the superclass *Person* has four attributes (*name, address, phone* and *dateOfBirth*) and the subclasses have attributes that are specific for only them. All objects of a subclass will have all attributes of its subclass and all attributes of the superclass. This is called *inheritance* and the subclass objects are said to *inherit* the attributes of their superclass object. In addition to attributes, the subclass objects are inheriting also all associations where the superclass is participating. Generally, it is rec-

ommended that the superclass should not have any objects on its own but all the objects should belong to some of the subclasses. [1, pp. 101–122] [7, pp. 79–106]

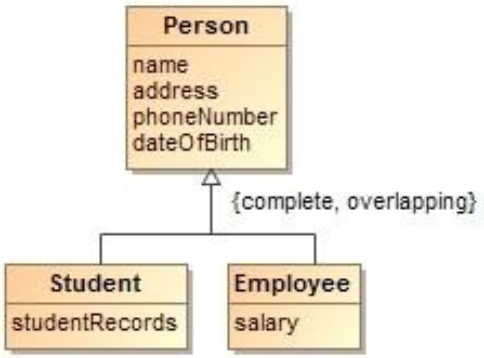


Figure 3. *UML class diagram representing specialization/generalization with complete and overlapping constraints. The model describes an imaginary University Database where a person can be either student, employee, or both.*

UML notation to model specialization is to draw a triangular end line from the subclasses to the superclass with combined lines as illustrated in Figure 3. The triangular end of the lines points towards the superclass. Optionally, there can be text in curly brackets ({…}) describing the four constraints that specify the characteristics of subclasses: If the subclass objects can exist only in one subclass of the specialization, the specialization is said to have *a* {*disjoint*} *constraint*. In our university database example, there can be a case that a person is simultaneously both an employee and a student at the university. In this case, when the subclass object can exist in multiple subclasses of the specialization, the specialization is said to have *an* {*overlapping*} *constraint.* Two more constraints are a {*complete*} and an {*incomplete*}constraint that describe whether or not the model includes all possible subclasses of the problem domain. In Figure 3, the model has a complete constraint because, in the problem domain of our university database, there are only two types of persons, students and employees, and the subclasses include them both. [1, pp. 101–122] [7: pp. 85–86]

Generalization is the reverse process for specialization. For example, if we first had two classes, *Student* and *Employee*, and we noticed that even though they are different classes, they still possess many common attributes such as *name*, *address*, and *phoneNumber*. Then we can use generalization to create a superclass to them holding the common attributes. Both generalization and specialization can lead to the same model. When modeling a conceptual schema using specialization, the method is called *a top-down refinement process* and when generalization is used, the process is called *a bottom-up conceptual synthesis*. In practice, it is likely that both of these methods are used in combination. [1, pp. 104–110] When one is thinking whether specialization or generalization can be used, the following two questions introduced by Clare Churcher [7, pp. 95] can be helpful:

- *"Do the two classes have enough in common to reconsider how they are defined?"* and
- *"Are some of the objects in a given class different enough from other objects to warrant reconsidering how they are defined?"*.

If the answer to the former question is yes, consider using generalization, and if the answer to the latter question is yes, consider using specialization. The drawbacks of these both methods are that they increase the complexity of the model. [7, pp. 112]

2.3 The Relational Data Model and Relational Database Constraints

The relational model was first introduced by Ted Codd working for IBM in 1970. It is based on the mathematical set theory and first-order predicate logic. The first commercial products based on the relational model appeared in the beginning of the 1980s. Before that, the popular models used in database products were the hierarchical and network models. In this section, we discuss about the relational model and relational database constraints. [1, pp. 141–142]

2.3.1 Relational Data Model

The relational data model represents a database as a collection of relations. *Relations* are usually represented as a table data values. Rows in a table are called *tuples* and columns are called *attributes*. A relation is a *set* of tuples. Each tuple represents related data values and corresponds to an entity or relationship in the data model terminology. All data in the same column have the same data type describing the types of values that the data can get. The set of all legal values for the attributes is called the *domain*. The domain is a constraint determining the logical definition, data type, and format of the attribute values. [1, pp. 142–143]

The logical definition defines, for example, that an attribute *mobilePhoneNumber* must be a 12-digit number representing a valid Finnish mobile phone number including a country code. *The data type* can define that the *mobilePhoneNumber* is of a string datatype. The *format* can define that the string representing the phone number has format *(+###) ### ### ###*, where each # represents a number 0–9. In addition to these, the domain can also define some other constraints such as unit of measurement. For example, it can define that the values in the attribute *personsHeight* represent the height in centimeters. [1, pp. 143]

A relation schema describes a relation. It can be described as $R(A_1, A_2, \ldots, A_n)$, where R is the relation name and A_i, $1 < i < n$, is the attribute of a relation. The number of attributes denoted as n is called *the degree* of a relation. The domain of an attribute A_i can be represented as $dom(A_i)$. *Relation* of the relation schema is called sometimes also

a relation state and is denoted as *r* or *r(R)*. A relation is a set of n-tuples $r = \{t_1, t_2, \ldots, t_m\}$ and each n-tuple is an ordered list of n values $t = <v_1, v_2, \ldots, v_n>$, where each value v_i, $1 < i < n$, is an element of $dom(A_i)$ or is a *null* value meaning that the value is unknown or does not exist. When we want to refer to the data value of the attribute A_i in tuple *t*, we can use notation $t[A_i]$. [1, pp. 143–147] Figure4 represents a relation *Person* corresponding to the class *Person* in Figure 3. The relation holds some example values for four different persons.

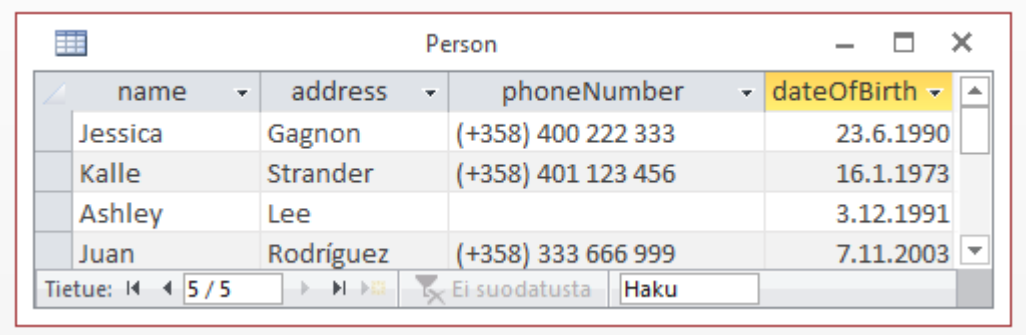


Figure 4. *Relation "Person" represented as a table. The relation has four tuples as four distinct rows and four attributes, one in each column heading.*

2.3.2 Relational Database Constraints

A relational database usually consists of multiple relations having multiple related tuples. The state of the database is a state of all relations forming the database. There are usually many constraints derived from the miniworld that restrict the values that can exist in a valid database state. These constraints can be divided into three categories: implicit constraints, explicit constraints, and business rules. The implicit constraints are constraints that are inherent to the data model described in the Section 2.3.1. For example, an implicit constraint defines that relation is a set of tuples meaning that a relation cannot have two identical tuples because mathematically a set does not include duplicate values. Business rules are constraints that are difficult to define in the data model and they are usually defined in the application programs. One example of a business rule is that a person's age must be between 15 and 75. In this section, we are interested in the explicit constraints only. They can be defined in the schema of a relational model by using the DDL (the DDL is defined in section 2.1.3). [1, pp. 149–150]

There is an inherent constraint defining that a relation cannot have two identical tuples but usually we have also a constraint that some subset of attributes cannot have identical attribute values within two distinct tuples. Being *SK* this subset of attributes and t_i and t_j are two distinct tuples within a relational state *r(R)*. We can define that

$$\forall t_i, t_j \in r, i \neq j, SK \in R : t_i[SK] \neq t_j[SK]. \quad (1)$$

Formula (1) defines *a uniqueness constraint*. The set of attributes *SK* are called *a superkey* of a relation schema *R*. A superkey can have attributes that have the same value in more than one tuple but at least one of the attributes in a superkey must be unique for all tuples in the same relation. Every relation must have at least one superkey, which is the combination of its all attributes. A definition that is more important than a superkey is a definition of *a key* that is specified to fulfill two conditions. First, a key must be a superkey and thus fulfill the uniqueness constraint specified in the formula (1). Second, a key must be *a minimal* superkey, which means that if we remove any attribute of the key, it does not satisfy the formula (1) anymore. A key that is formed of more than one attribute is called *a composite key*. If a relation schema has more than one key, each of the keys is called *a candidate key*. In Figure 5, there is a relation *Car* that has two candidate keys, *RegistrationNo* and *EngineNo***,** because both are unique for each car. We can then choose any of these candidate keys to be a primary key of the relation schema. *The primary key* is an attribute (or a set of attributes) that is used to identify each tuple in a relation. Because the primary key is used for identification, there is a constraint called *an entity integrity constraint*, which restricts that the primary key can never have *NULL* value. [1, pp. 66, 150–154]

Because a primary key identifies each tuple, we can use it to define relationships between relations. For this, we need a new key called *a foreign key*, which is a set of attributes *FK* in one relation that refer to the primary key attributes *PK* in another relation. Figure 5 illustrates two relations, *Person* and *Car*, where persons are related to cars as car owners by using foreign keys. Relation *Car* has a foreign key attribute *ownerSocialSecurityNo*, which relates each tuple in relation *Car* to the primary key attribute *socialSecurityNumber* in relation *Person*. [1, pp. 154] [8, pp. 126–128]

Person

socialSecurityNo	firstName	surName
230690-123X	Jessica	Gagnon
160173-1111	Kalle	Strander
021291-999S	Ashley	Lee
071103A4447	Juan	Rodrïguez

Car

RegistrationNo	engineNo	brand	ownerSocialSecurityNo
ABC-123	1110765	Volkswagen	071103A4447
QWE-333	4548805	Toyota	
LEE-11	2354409	Ford	021291-999S
SXV-959	5443766	Seat	160173-1111
GYL-321	1133003	Seat	230690-123X

Figure 5. *Two relations "Person" and "Car" that are related because each car can be owned by a person. This relationship is determined by a foreign key "ownerSocialSecurityNumber" that refers to the primary key "socialSecurityNumber" in relation "Person". The primary keys fields in both relations are shown with grey background color.*

In order to remain consistency between two related relations, we have one more constraint called *a referential integrity constraint*, which states that each foreign key referring to the primary key of another relation must always refer to an *existing* tuple in that relation. Being R_1 and R_2 two relation schemas, and *FK* is a set of foreign key attributes in R_1 that refer to *PK*, which is a set of primary key attributes in the relation R_2.

Formally, the referential integrity constraint is specified by the following two conditions:

1. The foreign key attributes *FK* must have the same domain as the primary key attributes *PK*, that is, *dom(FK) = dom(PK).*
2. The values of the foreign key attributes *FK* in the relation state $r_1(R_1)$ either equal the values of the primary key attributes *PK* in the relation state $r_2(R_2)$ or are *NULL*. In the former case, we have $t_1[FK] = t_2[PK]$, where t_1 is a tuple in r_1 relating to tuple t_2 in r_2.

All constraints mentioned in this section must be specified as part of the relational database schema and most commercial DBMS products offer features to accomplish this. [1, pp. 154–156]

2.4 SQL

Structured Query Language (SQL) is a standard query language for relational databases. It provides statements for data definition, queries and updates. In addition, it offers many other features, such as, making complex calculations, or specifying security and authorization. SQL is based on the greater extent on tuple relational calculus but has also features from relational algebra. However, compared to these two formal languages, SQL is more comprehensive, user-friendly, and expressive language. It is also relationally complete language meaning that all operations that are possible with relational algebra are possible also with SQL. [1, pp. 206, 233–234] [5] In following sections, we discuss about the history of SQL, and some basic queries that you can create with SQL.

2.4.1 The History of SQL in Brief

SQL is the standard of the International Standards Organization (ISO). The standard is called ISO/IEC 9075. [6] Originally, the SQL was developed by IBM Research. They created the early versions of this language calling it SEQUEL (Structured English Query Language). The first standard version of SQL was published in 1986 and it is called SQL1 (or SQL-86). The first version includes two levels: the first level is the core of the language and the second level includes some additional features. In 1992, the next version called SQL2 was released. It has three levels: the first one is called an entry level and it includes the both levels of SQL1 with some additional features; the second one is called an intermediate level; and the third one is called a full level.

In current commercial products, all platforms supporting SQL support at least the entry level of SQL2 and most often also the intermediate level. However, there has never been a platform fully supporting all features of the full level. There have been many updates to the SQL since SQL2 was first released. [1, pp. 234] [5] The latest update to

SQL standard is from 2011. [6] Figure 6 illustrates the various levels of SQL standard from the first version to the latest.

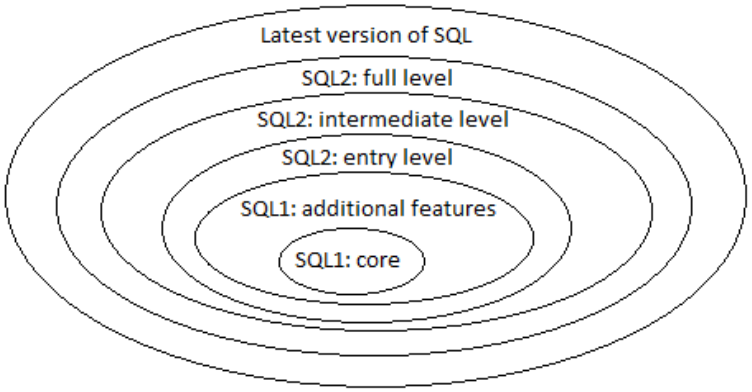


Figure 6. *Different levels of the SQL standard from the first version in 1986 (SQL1) to the latest version that includes all previous versions. [5]*

2.4.2 Basic SQL queries

Basic SQL query statements are simple and their structure consists of the following three parts:

- `SELECT` part determines which attributes of all attributes participating to the query are shown in the results
- `FROM` determines the relations participating to the query
- `WHERE` determines which conditions the results must fulfill [5]

A simple query example of an SQL statement for a relation *Employee(Name, Age, Address, ZipCode, City, Phone)* could be:

```
SELECT Name, Age
FROM Employee
WHERE Age > 25
```

This query above would return the name and age of all tuples from the relation *Employee* where the age is greater than twenty-five. According to the relational theory, a relation is a set of tuples (a set excludes duplicate elements) but in SQL the query results, which are relations, can include duplicate tuples. This is a useful feature if you, for example, need to calculate the number of the query results. In case, user wants to exclude the duplicates from the results, it is possible by adding word *DISTINCT* after *SELECT*. [5] The following query would return list of cities that begin by letter A, and each city would appear only once in the results even if they appeared in multiple tuples in the relation. The *Like* operator works as the equal (=) operator, and the asterisk (*) represents a wildcard character.

```
SELECT DISTINCT City
FROM Employee
WHERE City Like “A*”
```

Sorting data in SQL queries is possible with a clause *ORDER BY*. The default sorting order is ascending but if you want to sort in descending order, you need to use reserved word *DESC*. [5] Here is an example of an SQL statement that retrieves all fields from Employee table where the city is Helsinki. The results will appear sorted primarily by age in descending order and secondarily by name in ascending order.

```
SELECT *
FROM Employee
WHERE City = “Helsinki”
ORDER BY Age DESC, Name
```

In case some value in a tuple is missing (because it is not known or does not exist), the value is called *null*. This is not same as zero or an empty string. Null means that the value is missing completely. If there is a null value in the tuple, we cannot use normal comparison operators for these values in *WHERE* clause. For null values, there is a special operator *IS NULL*. [5] In the following query, we retrieve the name of all employees whose phone number is not recorded to the *Employee* table.

```
SELECT Name
FROM Employee
WHERE PhoneNumber IS NULL
```

There are many textbooks and other sources describing SQL statements in more detail. The purpose of this section was only to introduce the basics of the SQL. You can find more information about the SQL statements, for example, in *Access 2013 Bible* (see reference [2, pp. 407–425]).

2.5 Database Design Process

This section introduces a systematic database design process, also called as *design methodology*. This method includes a set of techniques that a designer can follow systematically. This can minimize the missteps in the design process and thus increase productivity. The larger and the more complex the database schema is, the more important a good systematic approach is for achieving a well-designed database in efficient manner. [1, pp. 403–404] [9, pp. 27]

In the first section, we look at the database design process as a whole. Then, the rest of the sections introduce us the most important steps of the design process in more detail giving instructions for creating a database from the initial requirements to a final tested application.

2.5.1 Database Design Process Models

The following sections introduce database design processes based on models from two different sources. We then use these models to introduce a customized design process model for the work in this thesis.

DATABASE DESIGN PROCESS BY ELMASRI AND NAVATHE

Authors Elmasri and Navathe introduce a database design process in their book Fundamentals of Database Systems (2007). Their process of designing a database application includes the following eight steps:

1. System definition
2. Database design
3. Database implementation
4. Loading or data conversion
5. Application conversion
6. Testing and validation
7. Operation
8. Monitoring and maintenance.

The first phase defines the scope of the database system. The second phase consists of designing the database from the requirements to the ready design that can be implemented on the chosen DBMS. In the third phase, the design is implemented by creating software applications and empty database files. Then in the fourth phase, the database is populated with data either directly or by converting them into correct format. If there are any software applications from the previous database application that must be converted to the new system, this is done in the phase five. The testing of the new system takes place in the sixth phase, and finally in the phases seven and eight, the new system is put into operation and maintenance of the system will continue through its whole lifetime. [1, pp. 408] Phases two and three, which form the database design and implementation phases, Elmasri and Navathe describe in more detailed with the following six steps:

1. Requirements collection and analysis
2. Conceptual database design
3. Choice of a DBMS
4. Logical database design
5. Physical database design
6. Database system implementation and tuning.

Requirements collection and analysis includes among others determining the users for the application, analyzing their needs, analyzing relevant existing documentation such as reports, and specifying the inputs and outputs for the transactions of the applica-

tion. Because these requirements are collected for an application that does not exist yet, they are likely to be incomplete but they will be transformed into more accurate specification in the next design phase. Requirements collection and analysis can be a time consuming phase but it is very important for the success of a design process with minimal amount of costly errors due to incomplete requirements. One way for gathering the initial requirements is to model them with the use case diagrams. These diagrams are one of the modeling languages offered by the UML (we discuss about use case diagrams in Section 2.5.2). [1, pp. 411–413, 435]

The conceptual database design phase consists of two parts: designing the conceptual schema and designing the transactions of the application. The conceptual schema is derived from the data requirements and it can be modeled using for example the UML class diagrams described in the Section 2.2. The aim is to keep the model as independent of any specific DBMS as possible even if the DBMS has already been chosen. The reasons for this are that the conceptual model is invaluable stable description of the data, the model provides a way for exact and straightforward communication, and the model helps achieving a complete understanding of the data. We have two ways to approach the process of transforming the requirements into a conceptual model. The first way is called *on shot*. In this approach, the requirements for different user groups or applications are first combined and then transformed into one complete conceptual schema. In the second approach called *view integration*, we first transforms the requirements of each user group or application into a conceptual schema and then combine the schemas into one global conceptual schema. The view integration approach is used mainly for large databases with many expected users. [1, pp. 413–416]

When transforming the requirements into a conceptual schema, we can use one of the four strategies: top-down strategy, bottom-up strategy, inside-out strategy, or mixed strategy. In *top-down strategy*, we first start with a higher-level abstraction and proceed into lower abstraction levels as the model develops. Specialization discussed in Section 2.2.4 is one example of this strategy. The *bottom-up strategy* is inverse to the top-down strategy and an example of this strategy is generalization discussed in Section 2.2.4. *Inside-out strategy* is a special case of bottom-up strategy where we first start from the most central concepts of the model and move outwards as the model develops. In the *mixed strategy*, we create some parts of the schema by using top-down strategy and some parts by using the bottom-up strategy. In the end, we combine the parts into schema. [1, pp. 415–416]

The second phase of the conceptual database design is designing the transactions of the application and it proceeds in parallel with the conceptual schema design. The goal of this phase is to determine the transactions in DBMS-independent way and ensure early on that all data that the transactions require are included in the conceptual schema. The transactions can be one of the three types: *retrieval transactions*, *update transaction*s, or *mixed transactions* of them both. Transaction design phase usually in-

cludes designing the inputs and outputs of the transactions and their functional behavior. [1, pp. 421–423]

After the conceptual design phase is completed, it is time to choose the DBMS that will be used for the database, unless the decision has already been made. Then, the logical database design phase begins including designing the conceptual and external database schemas for the chosen DBMS. In case of a relational database, the result of a data-model dependent conceptual schema design is a relational data model described in Section 2.3. [1, pp. 411, 426]

The logical database design phase is followed by the physical database design phase that includes designing the internal schema of the database. The internal schema design means designing physical storage structures and access paths for the new application. This is the design phase where you can affect the response time of transactions, the usage of storage space, and the transaction throughput (the average number of transactions that can be processed in one minute). [1, pp. 426–427]

Database system implementation and tuning is the design phase where the database and application programs are finally implemented, populated with data, and tested. The testing is done first individually for each transaction and application program and then for them all together. At this point, small design changes are still made. This is called database *tuning*. After the approved testing, the database application is deployed for service but the tuning continues through the whole lifetime of the application. [1, pp. 427–428]

DATABASE DESIGN PROCESS BY CLARE CHURCHER

Clare Churcher describes a database development process in her book Beginning Database Design (2007). The book introduces a diagrammatic notation of a software process model, which is based on the book Principles of Software Engineering and Design (1979) by Zelkowitz, Shaw, and Gannon. Figure 7 illustrates the model consisting of a square divided into four sections. The left half of the square describes the real world and the right half describes the abstract world. The design process starts from the upper left corner and continues clockwise until it reaches the lower left corner. During the development process, the problem is transferred from the *real world* to the *abstract world* via modeling. Abstraction helps achieving better understanding of the problem and possible solutions. After modeling, the application is designed. Then finally, the ready design is implemented, that is, the ready solution is returned from the abstract world to the real world. [8, pp. 11–12]

The first task in the square is the *problem statement*. In the end of this section, the initial requirements of the application are determined using for example UML use case diagrams. In the next phase called the analysis phase, some abstraction is needed to understand the problem more profoundly. We need to create the initial conceptual schema

and then compare it to the use cases. By asking questions about the use cases, we can understand the problem better. Then we can revise the conceptual schema and again compare it to the use cases. After several iterations, we will have the complete use cases

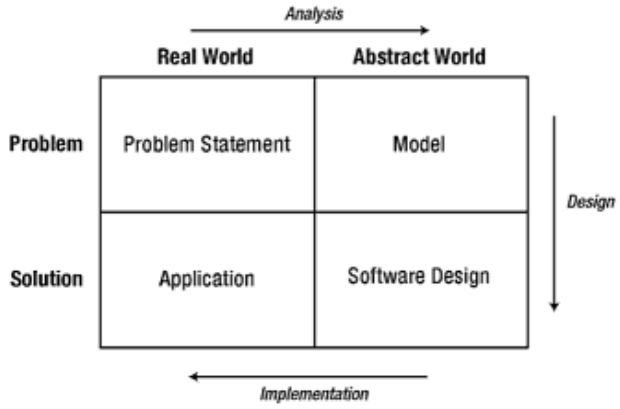


Figure 7. *The software design process based on the book Principles of Software Engineering and Design (1979) by Zelkowitz, Shaw, and Gannon. [8, pp. 12]*

and the conceptual schema (abstract model) of the database. In the design phase, the DBMS to be used is chosen and then, the conceptual schema is transferred to a relational schema. Finally, in the implementation phase, the design is implemented to the DBMS and the forms and reports are created and tested. [8, pp. 12–29]

DATABASE DESIGN PROCESS FOR THIS THESIS

In this thesis, the design process is following the steps described in Figure 8. This waterfall diagram combines the most relevant parts of the processes described earlier in this section.

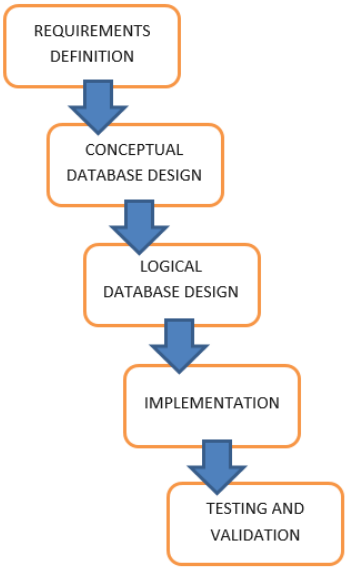


Figure 8. *The waterfall diagram describing the design process followed in this thesis.*

The following sections of this chapter describe in more detail the following two phases of the waterfall diagram: requirements definition (with UML use cases) and logical database design. The conceptual database design using UML class diagrams was already covered in Section 2.2. Microsoft Access, which is used for the implementation of the database application, is introduced in Section 2.6.

2.5.2 Requirements Definition and Analysis

In the requirements definition phase, our goal is to understand the problem completely before we start to solve it. We start by creating the initial use cases and then analyze all details, exception, irregularities, and possible uses of the system to see if our use case is describing the problem accurately. [8, pp. 31–32] In this section, we first look what use case diagrams are and then study how we can use them to define the requirements of a new application.

USE CASE DIAGRAMS

Use case diagrams are one of the many diagram types of the UML. Many projects begin with use cases because they help to visualize what is supposed to happen in the new system. Use cases diagrams are simple illustrations describing the interaction between the system and the actors. A use case is a sequence of events that result in some observable outcome for the actors. Actors can be users of the system or another system interacting with the system. In Figure 9, there is a typical use case diagram with two actors. The actors are represented as stick figures and the use case is represented as an ellipse. The actors have their role name written under them. A line is drawn between the actors and the use cases to represent the mutual relationship they have. One actor can be connected to many use cases and one use case can be connected to many actors.

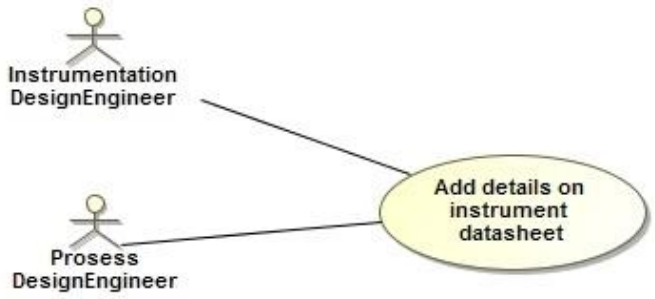


Figure 9. *A use case diagram representing two actors that are related to a common use case.*

The purpose of the use case is to provide a high-level model of what the system does and who are using it. These models can be then used in analysis, design, communication, and when creating test specifications. [7, pp. 20–24] Only the use case diagram is not enough to tell what the use case is about but also a short text chapter should be provided describing the sequence of events more specifically. [8, pp. 13] A *scenario* is a

description that determines all the possible courses that can be taken in the same use case. [7, pp. 21] For example in Figure 9, the process engineer might face one of the following scenarios:

- The process design engineer adds details on the datasheet successfully,
- The process design engineer decides to cancel the transaction, or
- The process design engineer enters an invalid value and gets an error message.

A scenario would describe the sequence of events in each of the cases above. A document template can be used in detailed descriptions. [7, pp. 25] In Appendix A, there is a use case template that is partially used in the use cases of this thesis. The template is downloaded from the website of IIBA (International Institute of Business Analysis) and it describes a sample use case of an ATM (Automated Teller Machine) transaction. [10]

When creating use cases, you can follow these four steps:

1. Find actors and use cases
2. Prioritize use cases
3. Develop each use case starting from the highest priority
4. Structure the use case model.

In the first step, you identify the actors and use cases by asking, who are the ones entering information to the system and who are the ones receiving information from the system. The purpose of the second step is to put the use cases in order starting from the highest priority so that most important use cases can be developed first in the third step. The use cases for the data entry have usually highest priority and the use cases for the data retrieval have the lowest. In the third step, the detailed description for each use case is created. This step can result also coming up with new use cases. Finally, the fourth step includes activities such as organizing use cases into packages, adding generalizations, and include and extend relationships. [7, pp. 31–34] Most of the activities of the fourth step are not described here in detail because the use cases of this thesis are so simple that the fourth step is practically excluded. However, include relationship is used in this thesis so it must be shortly explained.

Include relationship can be formed between two use cases. One of the use cases participating in this relationship is called an *included use case* and the other use case is called an *including use case*. Include relationship means that the behavior of the included use case is inserted into the behavior of the including use case. There are two cases where include relationship can be used. First, include relationship can be used to split complex use cases into simpler use cases. Second, if two or more use cases have common behaviors, these behaviors can be extracted into one separate use case. Include relationship is represented by drawing a dashed arrow between the use cases participating to the relationship so that the arrowhead is pointing to the included relationship. The arrow is labeled with the keyword «include». [15]

INITIAL INPUTS, OUTPUTS, AND DATA MODEL

In the requirements definition phase, we try to understand what tasks people using the system need to carry out and what data we need store to the database to support them in these tasks. Analyst must think the problem in the abstract level and define a model for the following three cases:

1. Input (use cases for the data entry)
2. Model (conceptual schema)
3. Output (use cases for the information output).

When defining the requirements, there are some questions which can provide a good start. By asking *"What does the user do?"*, the analyst can list all tasks that the user is regularly undertaking. When all tasks are listed, we can ask *"What data is involved?"*. This question helps the analyst determining what data needs to be recorded to the database in each of the tasks. *"What is the main objective of the system?"* is a question to define what tasks should be automated and what should be left to be done manually. It is important to keep the scope of the problem as small as possible in the early stages of the analysis. When the scope is defined, we can ask *"What data is needed to satisfy this objective?"* and define more specifically the tasks that the user undertakes. Then we can ask *"What are the input use cases?",* which means that we collect all tasks that include data entry and make the first input use cases from them. When we have some idea of the data entry tasks, we can ask the question *"What is the first data model?"* and create the first data model of the data needed for the data entry. The data model can be created using UML class diagrams. Finally, we can ask *"What are the output use cases?"*. This includes defining what data the user needs to retrieve from the database and create our first output use cases. [8, pp. 31–46]

Asking the questions of the previous chapter is an iterative process and can be repeated multiple times. Good sources for learning about the problem are interviews and existing documents such as forms and reports. Looking at the contents of the existing documents help to understand what kind of data we want to store to the database. [8, pp. 49–51]

2.5.3 Logical Database Design

Logical database design phase comprehends creating a conceptual schema and external schemas for the chosen DBMS. This is done by transforming the DBMS-independent conceptual schema created in the previous design phase into a relational schema. The transformation is called *mapping* and it results in DDL statements describing the relational schema of the database. [1, pp. 217, 426]

MAPPING

Mapping the conceptual schema into a relational schema can be done by following a five-step algorithm. The following paragraphs of this section describe each of these five steps in detail.

The first step comprehends *mapping of regular classes*. For each class in a UML class, we create a relation including the same attributes as the class. Then we choose one of the key attributes of the class as the primary key of the new relation. [1, pp. 219]

The second step comprehends *mapping of binary 1 to 1 associations*. There are several ways to map the binary 1:1 association but the most common is a foreign key approach. If we have two relations A and B, we choose one of them, for example, A and include as a foreign key in A the primary key of B. If the association has attributes, we include them into A. When choosing which one of the relations will be the one having the foreign key, it is better to choose the one that participates totally in the relationship because that way there will be less empty null values as the foreign key. [1, pp. 221] We can have for example a relation A representing ice hockey players, relation B representing ice hockey teams, and relationship R representing the relationship between the team from relation B and its captain from relation A. In this case, it is better to choose the relation B to include the primary key of A as a foreign key of B because all teams have a captain but not all players are a captain of some team.

The third step comprehends *mapping of binary 1 to N associations*. Mapping of binary 1:N association type works so that we choose the relation that represents the N-side of the association and include in it as a foreign key the primary key of the relation representing the 1-side of the association. If the association has attributes, we include them into the relation representing the N-side of the association. Again, also other approaches exist but the one introduced is by far the most common. [1, pp. 221–222]

The fourth step comprehends *mapping of binary M:N associations*. For each M:N association, we need to create new relation. Then we include as foreign keys in the new relation the primary key of both participating relations. The primary key of the new relation is a combination of both foreign keys. If the association has attributes, we include them into the new relation. [1, pp. 222]

The fifth step comprehends *mapping of specialization and generalization*. There are several ways to map data models where classes are participating into generalization/specialization. This section introduces four different techniques for mapping these models. Being C a superclass of m subclasses $\{S_1, S_2, \ldots, S_m\}$ and being $\{k, a_1, a_2, \ldots, a_n\}$ the attributes of class C where k is the primary key.

1. The first mapping technique is to create a relation L for superclass C and relation L_i for each subclass S_i, $1 \leq i \leq m$. L will have the same attributes as C and also the primary key of L will be k. The attributes of each L_i will be {attributes of S_i} ∪ {k} and the primary key of each S_i will also be k. Figure 10 (a) illustrates the result of this technique. This first mapping technique is suitable for all generalization/specialization models whether they are total or partial, or overlapping or disjoint.
2. In the second technique, the superclass C will not have an own relation. Only for each subclass S_i, $1 \leq i \leq m$, you need to create a be a relation L_i with attributes {k, a_1, a_2, ..., a_n} ∪ {attributes of S_i}. The primary key of each L_i will be k. This technique is illustrated in Figure 10 (b) and it is suitable for all generalization/specialization models except for those with *total* constraint.
3. The third technique is to create only one relation L with attributes {k, a_1, a_2, ..., a_n} ∪ {attributes of S_1} ∪ ... ∪ {attributes of Sm} ∪ {t}, where t is an attribute that determines the subclass to which each tuple belongs if any, and k is the primary key This technique is illustrated in Figure 10 (c) and it works for any generalization/specialization model except those with *overlapping* constraint.
4. In the fourth technique, you also create only one relation L but this time with attributes {k, a_1, a_2, ..., a_n} ∪ {attributes of S_1} ∪ ... ∪ {attributes of Sm} ∪ {$t1$, $t2$, ..., tm}, where t_i is an attribute with a Boolean value, which determines whether or not each tuple belongs to subclass S_i, $1 \leq i \leq m$, and k is the primary key This technique is illustrated in Figure 10 (d) and it works for any generalization/specialization model.
   [1, pp. 226–227]

The first technique works in all situations but has a drawback of creating relatively complex model with many relations. The second technique works well not only when the total but also when the disjoint constraint holds. With overlapping constraint, there will be redundant information because the attribute values of the superclass are stored in more than one relation L_i. One drawback with the model from the second technique is that every time we want to search for an arbitrary tuple of the superclass C, we must search from each relation L_i. The drawback of the models resulting from techniques three and four is that there will be a null value in each attribute of the subclass L_i to which the entity does not belong. It is recommended to prefer techniques three and four only when the subclasses have relatively small amount of attributes and prefer techniques one and two when the subclasses have relatively high number of attributes. [1, pp. 227–228]

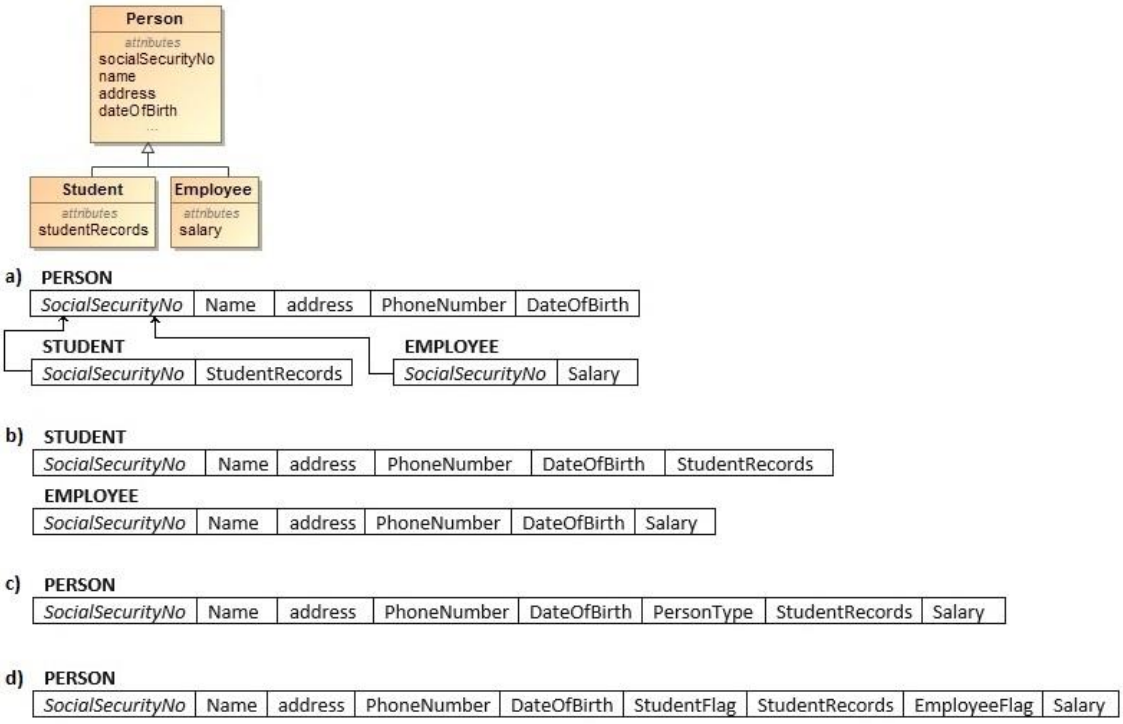


Figure 10. *Generalization model for classes Person, Student, and Employee and four alternative mapping techniques (a–d) for this model.*

FUNCTIONAL DEPENDENCY AND NORMALIZATION

Functional dependency is one of the most important concepts in the design of relational databases. [1, pp. 337] Functional dependency is used to define the concept of *normalization* which is again used for avoiding so called *update anomalies* in the relational model. Updated anomalies result from poorly designed relational model and they can lead to problems such as having inconsistent data due to repeated information, having difficulties to insert new records due to missing information in primary key fields, and loosing additional information when deleting a record. [8, pp. 142, 155]

Being A and B two attributes in a relation. If by knowing the value for A, we can uniquely define the value for B, we can say that attribute A *functionally determines* attribute B. For example, if A is person's social security number and B is person's name, we can uniquely define B by knowing A. If this is true for any values in A and B that the relation can ever have, we can say that A functionally determines B. Functional dependency can also be used to define the concept of primary key: primary key is a minimal set of attributes that functionally determine all other fields in the relation. [8, pp. 142–155]

Now, we can define also concept called *normalization*. There are several levels of normalization and these levels are called *normal forms*. Three first levels of normalization are usually enough for avoiding update anomalies. First normal form states that the primary key must functionally determine all other attributes in a relation. This practically means that an attribute value shall include only atomic data. [8, pp. 145–146] For

example, if a single attribute holds multiple values for person's phone numbers (work phone, cell phone etc.) and the primary key is person's social security number, by knowing the primary key we cannot *uniquely* determine person's phone number.

Second normal form states that the relation shall be in first normal form and all attributes shall be functionally dependent of the whole primary key. [8, pp. 148] For example, if two attributes, document number and document revision number, form the primary key and there is a third attribute, document title, in the same relation, the relation is not in the second normal form if the document number alone is enough to functionally determine the document title.

Third normal form states that a relation must be in first and second normal form and it cannot have attributes that are functionally dependent on any other attributes than the primary key. [8, pp. 149–150] For example, if a relation has three attributes: person's social security number (as primary key), person's passport number, and person name, the relation is not in third normal form because person's passport number functionally determines person's name even though the passport number is not the primary key. Condition for third normal form can be represented also in a single sentence: *a relation is in third normal form if all its attributes are functionally dependent on the primary key, the whole key, and nothing but the key*. Both normalization and conceptual database modeling lead to well-structured relational model. It depends always on the problem and the designer which one of these two methods should be used for achieving the best results. [8, pp. 151–153]

2.6 Microsoft Access

Access is a desktop database management system application from Microsoft. [17] The first version of the application was released in early 1990s. Access is an easy-to-use application providing both database engine and front-end design environment. This allows users to rapidly develop complete applications for their needs without need for traditional coding practices. [19] Access application has been designed so that it is a container of many different objects. There are six main types of these objects: tables, queries, forms, reports, macros, and VBA modules. These main objects again contain many other object types. [18] Without a few exceptions, all Access objects are located in a single Access file. [2, pp. 5] It is possible to create Access database applications by using only tables, queries, forms, and reports but with macros and VBA modules many things can be automated and applications can get plenty of added functionality.

2.6.1 Tables, Queries, Forms, and Reports

Access tables are the most important of all object types. [18] They are the containers for all data and data constraints stored into the application. [2, pp. 5] [18] Tables are equivalent to relations of the relational model. Each table holds data for a single entity,

such as instrument or cable. Tuples are the rows (called records) of Access tables and attributes are the columns (called fields) of Access tables. Tables are linked together by defining the relationships between them. [18] [2, pp. 5]

Access queries are used to retrive data from a database. With queries, user can retrieve a set of data that fulfill certain conditions. Most forms and reports are based on queries which filter and sort the data before it is displayed on the form or report. Queries can be also used to automatically modify, add or delete records on tables. Users can create queries either directly with SQL or by using a graphical query design tool which then transforms the query into SQL. [2, pp. 8]

Forms are the user interface in Access providing data-display and data-entry functions. [18] Forms enable more controlled and structured way to add or modify data in a database than what direct interaction with tables would. [2, pp. 8–9] Forms are built of objects called controls, such as *textbox*, *combobox*, *checkbox*, or *command button*. [18] When controls are added to forms (or reports), Access automatically gives them same name as the fields in recordset linked to the form (or report). This can cause errors because it is not possible to reference the control or linked field in the recordset individually if they do not have a unique name. In order to avoid this problem, it is better to give each control a unique name by adding a prefix to their names. There is a widely used standard called Reddick VBA (RVBA) Naming Conventions available in `www.xoc.net/standards`. The standard covers naming both Access and VBA objects. The standard was followed in this thesis to name the Access and VBA objects. Table 1 shows some examples about the naming conventions. [2, pp. 506, 672–673] [12]

Table 1. An example of standard RVBA naming conventions for some Access and VBA objects. [12]

Prefix	Object
tbl	Table (Access object)
qry	Query (Access object)
frm	Form (Access object)
sfr	SubForm (Access object)
rpt	Report (Access object)
bas	Module (Access object)
cbo	ComboBox (Access control)
txt	TextBox (Access control)
lbl	Label (Access control)
bool	Boolean (VBA datatype)
dbl	Double (VBA datatype)
int	Integer (VBA datatype)
str	String (VBA datatype)

There are at least three good benefits in using the naming conventions in VBA code. First, the naming convention helps in understanding the code because the datatype or object type will be clear from the name of the object or variable. Second, normally it is

not allowed to use reserved words such as *Print* in the code but with the naming convention it is possible to use for example a Boolean type variable name *boolPrin*t. Third, it becomes easier to debug the application code because it will be easier to notice if a variable has wrong scope or datatype. [2, pp. 770–772]

Access reports are used for processing and displaying data in nice format which can be then printed. As forms, also reports are built of various types of controls. Reports are based on queries and they provide very flexible way to represent data. Often reports combine data from many different tables which can have complex relationships between each other. [2, pp. 9] [18]

2.6.2 Access Programming with Macros and VBA Modules

Access macros are an easy and quick way to add functionality to Access application without a need to write VBA code. There are about 50 actions that macros can perform and many of these actions have parameters that user can specify. A macro can for example open a new form when a command button is pressed on another form. [18]

Modules are the containers for the VBA procedures. There are two types of modules in Access: *standard modules* and *class modules*. Standard modules are independent objects in Access in the same way as other objects such as forms and reports. Standard modules are supposed to contain procedures that are needed everywhere in the application because they can be accessed by any other module in Access. Class modules, on the other hand, are bound to forms and reports and cannot be accessed from anywhere else. Another difference between class modules and standard modules is that class modules support *event procedures*. The event procedures are triggered when the user for example clicks a button on a form resulting an event procedure to be called from the class module bound to the form. [2, pp. 722–723]

VBA provides a powerful and flexible way to manage the data in an Access application. Anything that can be done by using an access form, can be done also by VBA code. There are two different object models in Access that can be used to manipulate data with VBA code: ActiveX data object (ADO) and data access objects (DAO). The former is good for advanced data manipulation task when the latter is the best choice for all routine tasks. These two object models are owned by the Access database engine and are separate from Access object model including Access database objects. In this thesis, only DAO is used in VBA code. [2, pp. 855–856]

3. DESIGN AND IMPLEMENTATION

3.1 Requirements Definition

This chapter begins the development process of the database application for detailed instrumentation engineering. The development process continues until the end of this thesis. In this chapter, we define and analyze the requirements for the application. We start with requirements definition. After that, follows the conceptual database design phase, then logical database design phase, and finally implementation phase.

3.1.1 Scope Definition

The application was designed for the real-world project. The objective of the application was to provide a software tool for instrumentation engineering to store and analyze the design data and provide the needed reports. This section introduces two scopes: the scope of the case project from the instrumentation point of view and the scope of the application to be developed.

The case project was a big factory expansion project covering among others about thousand five hundred I/Os, almost thousand new instruments, the expansion of four control systems, and nearly hundred new electric motors. The process area of the factory was divided into ISBL (Inside Battery Limits) and OSBL (Outside Battery Limits) areas. Within these areas, the process was divided into more specific process areas, which were designated by two-letter descriptors. Table 2 illustrates the process areas covered by the case project and describes the control system controlling each of these areas.

Table 2. A list of process areas and related control systems in the case project.

Battery limit	Process area	Control system
ISBL	BF	ISBL DCS/PLC
ISBL	CT	ISBL DCS/PLC
ISBL	FT	ISBL DCS/PLC
ISBL	IP	ISBL DCS/PLC
ISBL	MT	ISBL DCS/PLC
ISBL	PT	ISBL DCS/PLC
OSBL	PH	OSBL PLC
OSBL	YD	OSBL DCS

In ISBL area, there were two control systems: ISBL DCS and ISBL PLC. ISBL PLC serves as SIS (Safety Instrumented System). In OSBL area, there was OSBL PLC controlling PH area and OSBL DCS controlling YD area.

The communication between the instruments and the control systems worked totally using conventional 0/24 V binary signals and 4–20 mA analog signals. The cabling infrastructure followed a principle that field cables from the instruments were collected to junction boxes in the field. Then trunk cables were installed from the junction boxes to the marshalling cabinets or directly to the I/O cabinets.

There were four different electric motor starting types that were used in the project:

- Direct On Line (DOL),
- Soft Starter (SS),
- Variable Frequency Drive (VFD), and
- 2-Speed Motor (2-S).

All motors were powered from the motor control center (MCC) regardless their starter type. The communication between the motors and the control system was done using the conventional 0/24 V binary signals and 4–20 mA analog signals. The signals were transmitted via the cables installed between the control system and the MCC. The amount of the signals depended on the motor drive type.

The database application must be capable of producing the following reports: instrument datasheets, instrument list, I/O list, cable list, equipment list, pipeline list, and motor list. The application must also be capable of saving the data needed in the reports and allow users to do the needed data analysis with the data.

There are four types of users that could use the database: instrumentation engineer, process engineer, electrical engineer, and database administrator (DBA). The instrumentation engineer is the main user to whom the application is designed but there will be some data that should be controlled solely by process engineer or electrical engineer. However, because the application is only a prototype, designing separate user interfaces or user profiles for different user groups is outside the scope of this thesis. The DBA is the person modifying the database structure and user interface when needed.

3.1.2 Instrument Datasheets

In the case project, the instrument datasheets were documents that contained detailed specification of each instruments on a single A4 size sheet. The requirements for the data needed in the application were determined by analyzing all existing datasheet templates. There were altogether twenty-eight different datasheet templates, one for each instrument type. The first step was to analyze all datasheet fields and see which of them

are common for all datasheets. Table 3 lists all twenty-eight instrument types and categorizes them first into sensors and valves, and then the sensors are further categorized according to the instrument type. The table shows also the instrument tag type that was used for each instrument type.

Table 3. A list of datasheet templates used in the case project.

Index	Datasheet template	Sensor/ Valve	Instrument type	Instrument tag type
1	Gas analyzer	Sensor	Analyzer	AT
2	Corrosion analyzer	Sensor	Analyzer	AT
3	Conductivity analyzer	Sensor	Analyzer	AT
4	Dew point analyzer	Sensor	Analyzer	AT
5	pH analyzer	Sensor	Analyzer	AT
6	Radioactive density analyzer	Sensor	Analyzer	AT
7	Annubar flowmeter	Sensor	Flow	FT
8	Differential-pressure wedge flowmeter	Sensor	Flow	FT
9	Integral orifice flowmeter	Sensor	Flow	FT
10	Magnetic flowmeter	Sensor	Flow	FT
11	Coriolis flowmeter	Sensor	Flow	FT
12	Vortex flowmeter	Sensor	Flow	FT
13	Rotameter	Sensor	Flow	FICV
14	Guided wave radar transmitter	Sensor	Level	LT
15	Radioactive level transmitter	Sensor	Level	LT
16	Radar level transmitter	Sensor	Level	LT
17	Level switch	Sensor	Level	LSH/LSL
18	Differential pressure transmitter	Sensor	Pressure	PDT
19	Pressure gauge	Sensor	Pressure	PI
20	Pressure transmitter	Sensor	Pressure	PT
21	Pressure switch	Sensor	Pressure	PSH/PSL
22	Bimetallic and glass thermometer	Sensor	Temperature	TI
23	Temperature switch	Sensor	Temperature	TSH/TSL
24	Temperature transmitter	Sensor	Temperature	TT
25	Control valve	Valve	N/A	CV
26	Block valve	Valve	N/A	HV
27	Pressure regulating valve	Valve	N/A	PCV
28	Pressure relief valve	Valve	N/A	PSV

All data fields in each of the twenty-eight datasheet templates were examined and written down on a spreadsheet. There were altogether almost three-hundred different data fields on these templates. The fields that were common for every datasheet template are listed in Table 4.

Table 4. A list of datasheet fields that were common for every datasheet template in the case project.

Index	Data field name	Remarks
1	Instrument tag	
2	Related process unit	
3	Client's purchase order code	
4	Client's document number	
5	Page number	
6	Name of the original project	
7	Revision history	many fields
8	Revision number for each data field	many fields
9	Manufacturer	
10	Model	
11	Manufacturer's order code	
12	Instrument type	
13	Service description	
14	P&ID document number	
15	P&ID revision number	
16	Hazardous area classification	classification for the area of use
17	Eqipment's Ex marking	classification for the equipment
18	Safety integrity level	
19	Power supply	
20	Related vessel/equipment tag	
21	Related pipe line tag	
22	Related pipe line material	
23	Process fluid composition	min/norm/max concentration
24	Process fluid name	
25	Process fluid phase	
26	Design pressure	
27	Design temperature	
28	Required certificates	
29	Comments/notes	many fields

By asking questions introduced in Section 2.5.2, we can come up with the initial use case for the datasheet application from the datasheet perspective. The use diagram for the datasheets is shown in Figure 11. The users for the datasheets are an instrumentation design engineer and a process design engineer. The main objective of the application is to allow the instrumentation design engineer to add and remove instruments. When the user adds an instrument to the database, it creates a new datasheet to the application. Instrumentation and process engineers have also common tasks that they can perform: add and modify instrument details, retrieve instrument information, and print instrument datasheet. The only differences between these tasks are that the instrumentation and the process engineers will maintain different data. Process engineers add and modify only process data, such as process fluid, design temperature, and design pressure. Instrumentation engineers modify everything else on the datasheets except process data. This way

the different disciplines will not have overlapping tasks and the responsible discipline is easy to distinguish. Retrieving instrument details and printing instrument datasheets are the output use cases and the rest are input use cases.

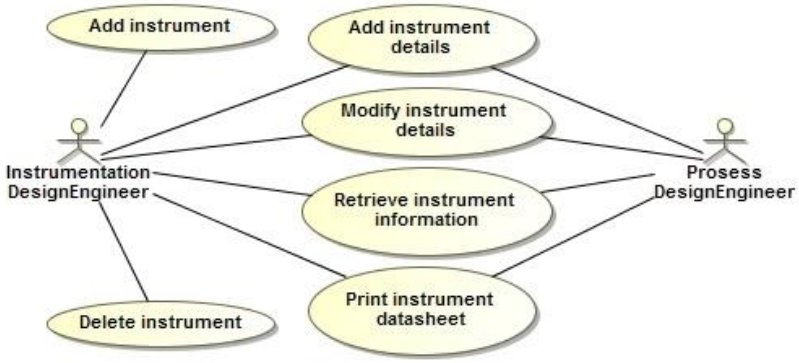


Figure 11. *The initial use cases of the database application from the instrument datasheet perspective.*

By thinking the data needed in the datasheets, we can come up with the initial data model shown in Figure 11. We begin with the top-down approach and model the first superclass, *Instrument*. Then we can use specialization to categorize the *Instrument* class into *Sensor* and *Valve* classes. These both classes inherit the common attributes for all instruments (see Table 4) from their superclass but the attributes belonging to *Sensor* and *Valve* class differ from each other.

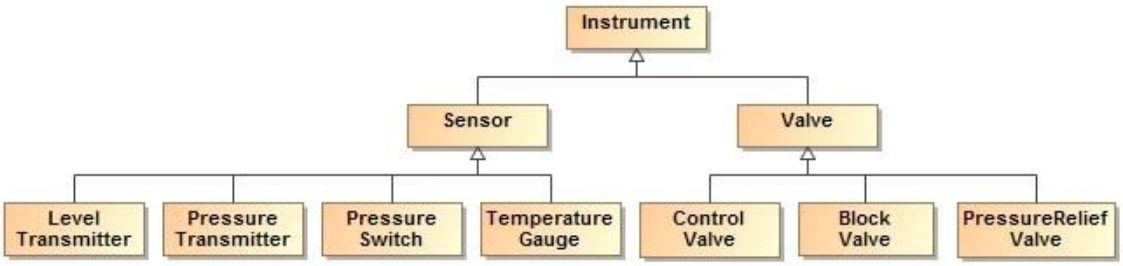


Figure 12. *The initial data model for the database application from the datasheet perspective. Instruments can be categorized into sensors and valves that can be further categorized into different instrument types. In order to save space, all instrument types are not modelled in the lowest hierarchy of this diagram.*

Specialization can be continued further to model *Sensor* class as a superclass for different instrument types such as level transmitter, pressure transmitter and pressure switch. The same can be done for the *Valve* class by modeling subclasses such as control valve and block valve that all inherit the common attributes for all valves from the superclass *Valve*. In order to keeps the diagram small enough, Figure 11 does not describe all twenty-eight different instrument types used in the case project.

3.1.3 Instrument List

The second report type to be analyzed was the instrument list. In the case project, the instrument list was a spreadsheet containing many columns of information. The first step in the analysis was the same as with the datasheets: to list all different data fields and see what data was common for all instruments. Table 5 shows the result of the analysis by listing the most important data fields of the instrument list. Some of the fields were redundant with the contents of the datasheets.

Table 5. The most important instrument list fields used in the case project.

Index	Data field name	Remarks
1	Battery limit	ISBL/OSBL
2	P&ID area	
3	P&ID number	
4	P&ID drawing number	from P&ID area and P&ID number
5	P&ID coordinate	
6	P&ID revision	
7	Instrument tag type	
8	Instrument tag area	
9	Instrument tag number	
10	Instrument tag	from instrument tag type, area, and no.
11	Instrument description	e.g. temperature transmitter
12	Remote transmitter	yes/no
13	Process equipment or pipeline	
14	Category	in-line/on-line/off-line
15	Service	
16	Location	e.g. field / MCC / control room
17	Status	e.g. new/existing/replaceable/modifiable
18	Package code	name of the package unit
19	Package vendor	vendor supplying the package unit
20	Instrument installer	e.g. instrument/piping contractor
21	Field cable installer	e.g. instrument contractor / vendor
22	Eqipment's Ex marking	classification for the equipment
23	Safety integrity level	e.g. none/SIL1/SIL2/SIL3
24	Power supply	e.g. external power supply / loop pow-
25	Instrument air manifold tag	
26	Instrument layout drawing number	
27	Instrument datasheet drawing no.	
28	Hook-up drawing numebr	
39	Manufacturer	
30	Model	
31	Procurement code	
32	Procurement status	e.g. inquired/quoted/purchased
33	Remarks	
34	Internal notes	
35	Deleted	true/false
36	Revision	

The initial use case diagram for the instrument list is not shown here because the use cases for the instrument list are very similar to the use cases of the datasheets. The inputs are exactly as in Figure 11 and the only difference in the outputs are that instead of printing the instrument datasheets, we print the instrument list report. The difference between the actors is that the only actor for the instrument list data is the instrument design engineer.

The class diagram for the instrument list data was formed by analyzing the data shown in Table 5. The class diagram is shown in Figure 13.

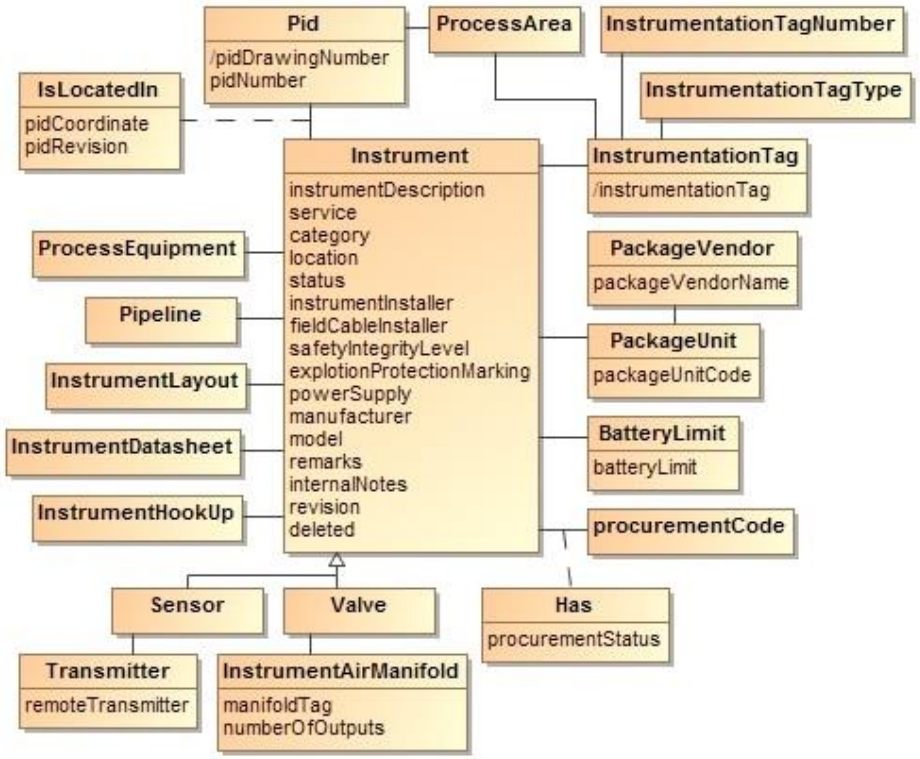


Figure 13. *The initial conceptual schema for the instrument list data.*

The central part of the class diagram is the *Instrument* class holding all attributes that describe instruments. *Process Area* class is related to two classes because the process area code is needed both in *Instrumentation Tag* and *Pid* classes. The association between *Pid* class and *Instrument* class has two attributes for P&ID coordinate where the instrument is located and also for the latest P&ID revision where the instrument existed. Also, the assoaciation between *ProcurementCode* class and *Instrument* class has one attribute for procurement status.

3.1.4 I/O List

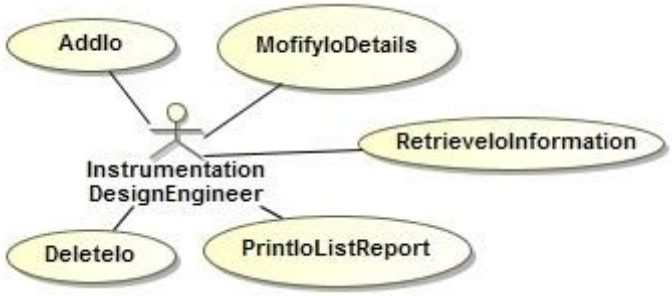
The I/O list is a document listing all inputs and output to the control systems. In the case project, there were four different control systems as shown in Table 6.

Table 6. The most important I/O list fields that were used in the case project.

Index	Data field name	Remarks
1	Battery limit	ISBL/OSBL
2	P&ID area	
3	P&ID number	
4	P&ID drawing number	from P&ID area and P&ID number
5	P&ID coordinate	
6	P&ID revision	
7	I/O tag type	
8	I/O tag area	
9	I/O tag number	
10	I/O tag	from I/O tag type, I/O area, and I/O no.
11	I/O description	e.g. temperature transmitter
12	Location	e.g. field / MCC / control room
13	Is motor signal	yes/now
14	Nominal current	Motor current
15	Motor tag	
16	Motor type	
17	I/O type ISBL DCS	AI/AO/DI/DO
18	I/O type ISBL safety PLC	AI/AO/DI/DO
19	I/O type OSBL DCS	AI/AO/DI/DO
20	I/O type OSBL PLC	AI/AO/DI/DO
21	Interlock DCS	interlock tag
22	Interlock safety PLC	interlock tag
23	Safety integrity level	
24	Measurement range lower limit	
25	Measurement range upper limit	
26	Measurement rage unit	
27	IntelockLimitValue	
28	IntelockDescription	
29	AlarmLimitValue	
30	AlarmDescription	
31	Controller type	single/cascaded/complex/other/none
32	Remarks	
33	Internal notes	
34	Deleted	
35	Revision	

The initial class diagrams model for the I/O list is modeled in Figure 14a and the conceptual schema for the I/O list is modeled in Figure 14b.

a)



b)

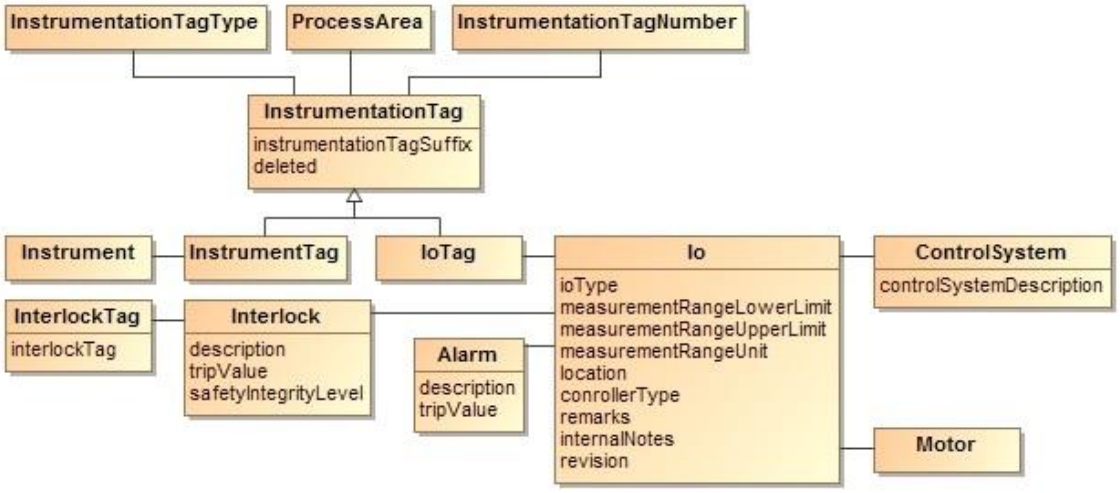


Figure 14. *The initial use case diagram for the I/O list data (a) and the initial conceptual schema for the I/O list data (b).*

In the class diagram model described in Figure 14b, *InstruemntationTag* class is holding all tags needed both for instruments and I/O. , *InstrumentationTag* class works as a superclass for subclasses and *IoTag* which are then related to *Io* and *Instrument* classes. Control systems, motors, alarms, and interlocks have their own classes having an association with *Io* class as they are all distinct entities of the miniworld.

3.1.5 Cable List

Cable list is a document that among others a contractor is using to install cables. In the case project, the cable list included the data fields shown in Table 7. When contractor pull the cable, they need to know which cable (cable tag) is to be pulled and where the cable ends are to be connected (from/to). Cable description provides information about the usage of the cable. This is useful information, not just for the contractor, but also for

an engineer or a client reading the cable list. Cable-length field is needed to purchasing correct amount of cable. Cable type and size information are needed in purchasing as well as in pulling the correct cable type. Cable ID field is important in identifying the cable even if its cable tag must be changed for some reason. Remarks field provides a place for additional information. If the contractor or client receives several revisions of the cable list, revision field indicates the latest revision number for each cable. Revision history fields are in the beginning of the cable list report including information such as revision number, date, responsible person, person who checked the document, person who approved the document and revision description (for comments, approved for construction, as built etc.).

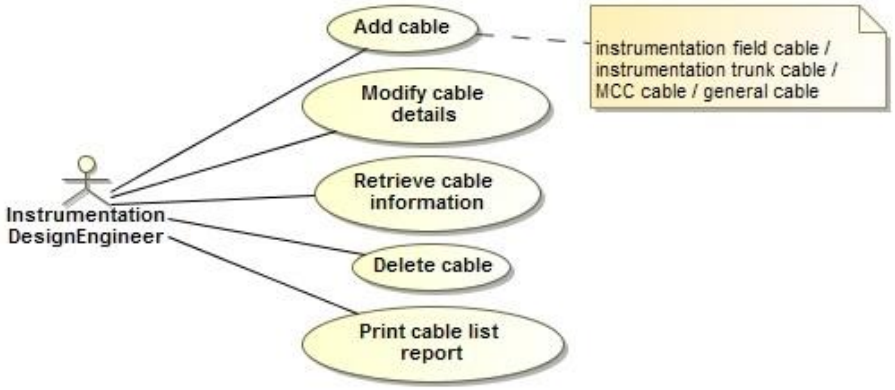
Table 7. A list of data fields that were used in the cable list report of the case project.

Index	Data field name	Remarks
1	Cable ID	
2	Cable tag	
3	Cable description	
4	From	
5	To	
6	Cable length	
7	Remars	
8	Revision	
9	Revision history	many fields

From the instrumentation point of view, we can categorize the cables used in the case project into four categories: instrumentation field cables (from instruments to the junction boxes), instrumentation trunk cables (from junction boxes to marshalling/DCS/PLC cabinets), motor signal cables (from marshalling/DCS/PLC cabinets to MCC), and all other cables (general cables), which do not fit to any of the previous categories. Using generalization, we end up to a hierarchal class diagram model described in Figure 15b. Class *Cable* is a superclass for the four different cable categories. Instrument field cables are connected to instruments in one end and to junction boxes in the other end. Instrument trunk cables bring the signals from junction boxes to either marshalling cabinets or control cabinets. Motor cables belong to class *MccCable* and they are connected from motors to either marshalling cabinets or control cabinets, as well. *GeneralCable* class holds all other cables that can be connected between any two points at the factory. Both junction boxes and all cabinets are classified to be part of class *ElectricalEnclosure.* The terminal rows of these electrical enclosures are different objects from the enclosures so they are modeled to be part of class *ElectricalEnclosureTerminalRow.* The cable types of three different categories out of four have then associations either to *Instrument* class or *ElectricalEnclosureTerminalRow* class. General cables do not have associations to these classes because they must have freedom to be

connected to any point at the factory so their connection points are recorded to the attributes *from* and *to* of GeneralCable class. Cable types are also objects as their own so they are modeled to have class *CableType* which has an association with *Cable* class.

a)



b)

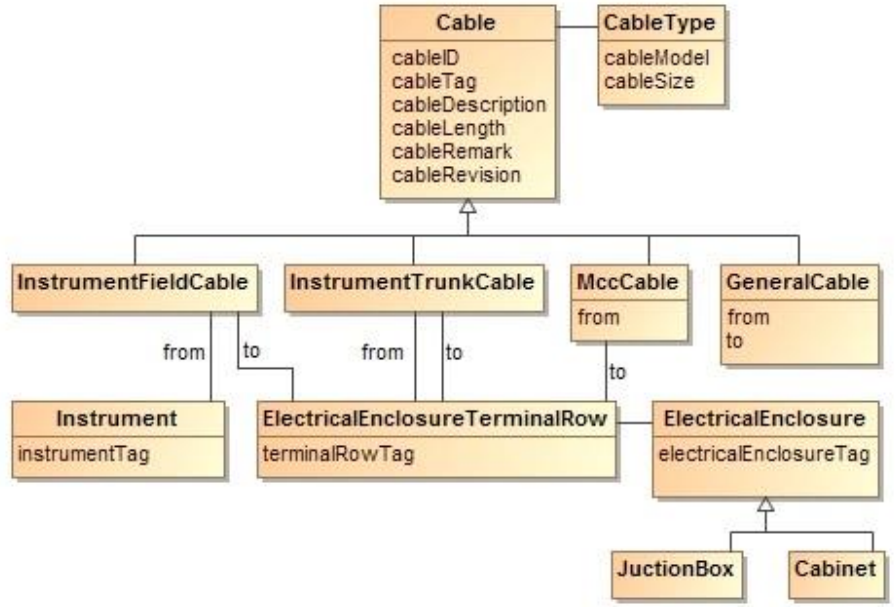


Figure 15. *The initial use case diagram for the cable list data (a) and the initial conceptual schema for the cable list data (b).*

The initial use case diagram is modeled in Figure 15 a. The only actor participating to this use case is the instrumentation engineer, and the use cases are similar to other use cases modeled so far. *Add cable* use case has a note indicating all four different cable types that the actor might need to add to the database.

3.1.6 Equipment List

Equipment list is a document, which the process department used to list all new process equipment in the project. The list included several columns of useful information for the process designers. The most important columns for the instrumentation department are listed in Table 8. These columns represent the data that the database application should be capable of storing. All possible columns used in the project are not used in the database application because the main goal of the application is to serve the instrumentation engineers and it is important to keep the scope of the application as small as possible in order to avoid too complicated application design.

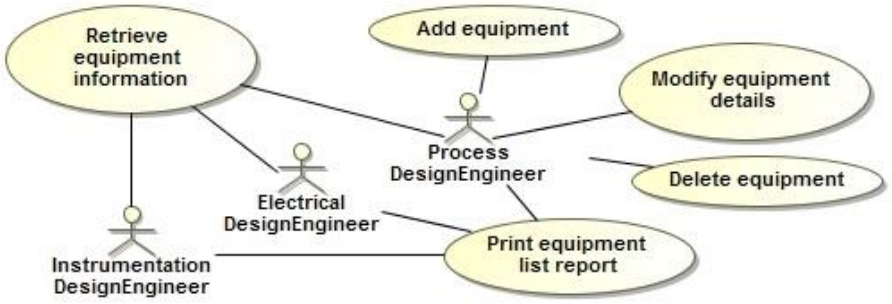
Table 8. A list of data fields that were used in the equipment list report in the case project.

Index	Data field name	Remarks
1	Equipment number	unique number for each equipment of same type
2	Equipment description	
3	Design temperature	
4	Design pressure	
5	Operating temperature	
6	Operating pressure	
7	Remarks	remarks that are shown in a published report
8	Internal notes	remarks for internal use only
9	Process area	
10	Equipment type description	e.g. heat exchanger
11	Equipment type symbol	e.g. “E” for a heat exchanger
12	Related PID	Piping and Instrumentation Diagram
13	Related PFD	Process Flow Diagram

The application user interface could be designed so that only the process department has access to modify this data and the other disciplines should have access to only view this data. However, this functionality is out of the scope of this thesis. Instrumentation engineers need process equipment data, for example, for marking equipment on which some instrument is installed. Electrical department need the data to link the electric motors to the related process equipment, such as a pump or agitator.

Figure 16a shows the initial use case for the equipment list data. The use case diagram shows the inputs and outputs of the database application for the equipment data. Inputs include the process engineer adding and deleting equipment and modifying equipment details. Outputs are a process, electrical or instrumentation engineer retrieving equipment information and printing the equipment list report. Using the data in table 8, we can create the first conceptual schema describing the data structure of the equipment data. The initial conceptual schema for this data is shown in Figure 16b.

a)



b)

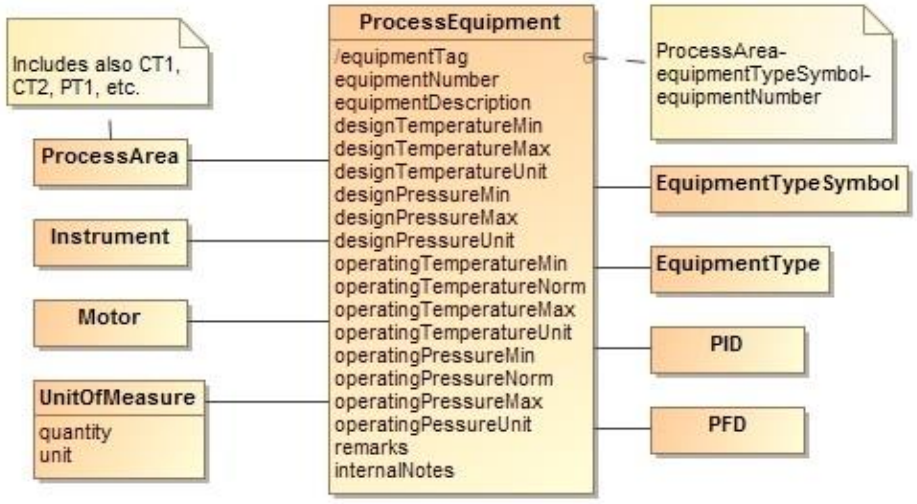


Figure 16. *The initial use case for the equipment list data (a) and the initial conceptual schema for the equipment list data (b).*

The first attribute of the *ProcessEquipment* class is *equipmentTag*, which is a calculated attribute representing a unique tag for each process equipment. The note attached to the attribute describes that the equipment tag is calculated from three different attributes. The *ProcessArea* class has also a note describing that the process area includes not just process areas described in Table 1 but also process subareas. In the case project, each process area can be divided into subareas that are designated with a number (1 or 2) immediately after the two-letter process area acronym. For example, the CT area can be divided into CT1 and CT2 areas. Some process areas do not have subareas, and in that case, the process area can be described with the equivalent designation by adding simply a number "1" after the process area, as for example PH1. Design pressure and temperature have minimum and maximum values which should be stored separately as a number datatype. Also, the unit of measure should be stored separately as a text datatype. The operating temperature and pressure values have also a normal value in addition to the values that the design parameters have. The measurement values are associated to the class *UnitOfMeasure* including all measurement units needed in the da-

tabase application. *Instrument* and *Motor* classes are associated with the process equipment to which they are installed.

3.1.7 Pipeline List

The pipeline list is a document controlled by the process department. The document lists all new pipelines that are needed in the process. It might include also existing pipelines if they are to be demolished or modified. The instrumentation department uses the pipeline list to link instruments to the related pipelines and to know details about the pipelines such as nominal diameter, material and pressure class. In the case project, the most important pipeline list fields for the instrumentation department are shown in the Table 9.

Table 9. The most important pipeline list fields for the instrumentation department used in the case project.

Index	Data field name	Remarks
1	Pipeline number	unique four digit number for each pipeline
2	Process area	according to the Table 1
3	Process fluid code	two letter designation describing the fluid
4	Process fluid	
5	Pipeline diameter	single value or two values (min. and max diameter)
5	Pipeline insulation	
6	Pipeline tracing	heat tracing type for the pipeline
7	Pipeline specification	
8	Pipeline material	
9	Pressure class	
10	P&ID from	PI&D from which the pipeline starts
11	P&ID to	P&ID to which the pipeline ends
12	Status	existing/new line
13	Design pressure	minimum and maximum values
14	Design temperature	minimum and maximum values
15	Operating pressure	normal value
16	Operating temperature	normal value
17	Remarks	general comments
18	Revision	

The pipeline list is modified by the process department and only viewed by the instrumentation department. Pipeline list is one of the main documents for piping department but it is also important especially for instrumentation department. Figure 17 shows the initial use case model for the pipeline list data.

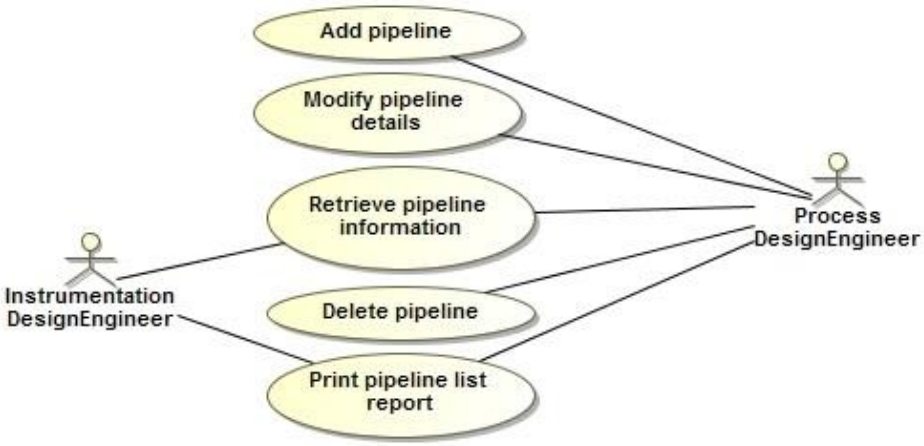


Figure 17. *The initial use case for the pipeline list data.*

In the case project, each pipeline had a tag that consisted of several parts separated by a dash symbol. In general, a pipeline tag had the following form:

AABB-CCCC-D"-E-F"-G,
where
AA is the two-letter process area code as shown in Table 1,
BB is a two-letter process fluid code described with two letters,
CCCC is a four-digit pipeline number being unique for each pipeline,
D" is a number describing the pipe nominal diameter in inches or in millimeters,
E is the pipeline specification code describing the pressure class and material,
F is a code describing the pipeline insulation material and thickness in inches, and
G is a one or two-digit number describing the heat tracing temperature in °C.

Each pipeline having a unique pipeline number can go through several process areas or transfer several process fluids in different parts of the pipeline. In the case project, the combination of the process area code, process fluid code, and pipeline number formed one row in the pipeline list. There was no more than one row with the same pipeline designation *AABB-CCCC* on the list. On the pipeline having the same process area, process fluid, and pipeline number, there could still be a variety of pipeline nominal diameter, material or pressure class. On the pipeline list, these multiple values were shown so that if the nominal diameter varied on the pipeline, the list showed the minimum and maximum diameters. In case of the multiple material or pressure class, the multiple values were listed after each other and separated by a forward slash symbol (/). There were no rows having multiple values for the insulation or heat tracing. We can define that if such rare case is encountered that the same pipeline row on the pipeline list should have several values for either the insulation or heat tracing specification, then this exception can be described for example in the *remarks* field.

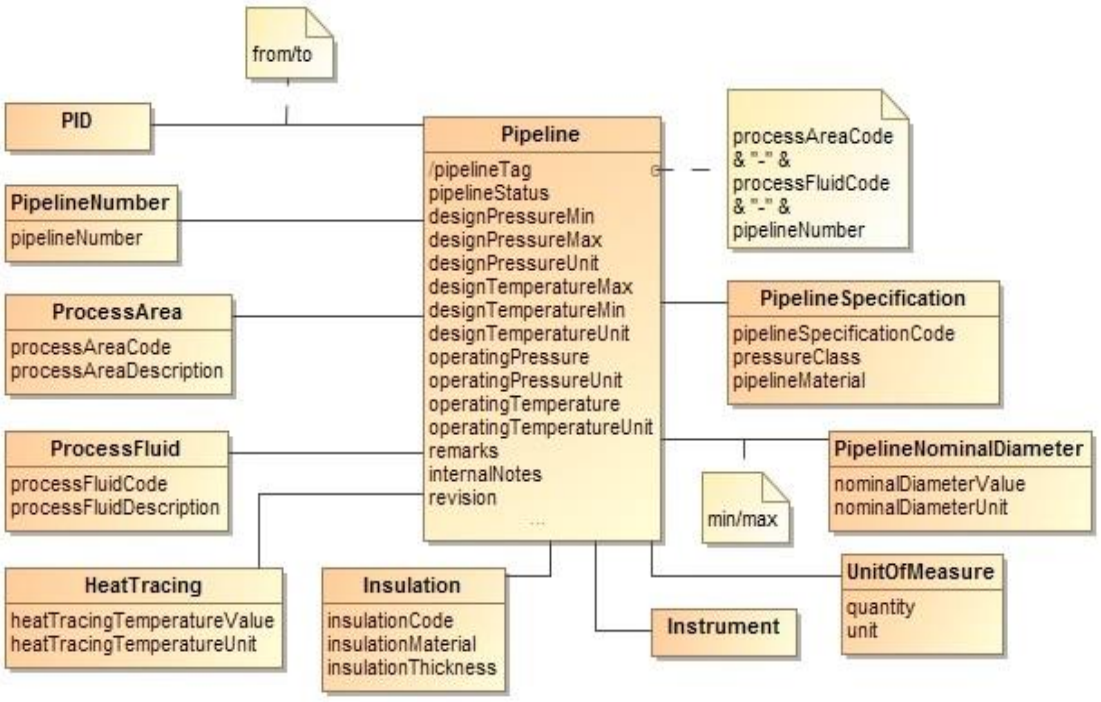


Figure 18. *The initial conceptual schema for the pipeline list data.*

Figure 18, shows the initial conceptual schema for the pipeline list data. The *Pipeline* class has an attribute *pipelineTag*, which is a calculated attribute and its value must be unique for each object of the class. The attribute *status* is a Boolean datatype attribute having either value existing or new. The operating and design conditions will use the unit of measure from the class *UnitOfMeasure.* The *Instrument* class represents instruments that are installed in the pipeline form the *Pipeline* class. Heat tracing and insulation are as own classes so that it is easier to keep the markings consistent and still be able to easily add new heat tracing or insulation types. *ProcessFluid* class includes both the *processFluidCode* and *processFluidDescription* attributes. In the case project, multiple process fluids can have the same process fluid code. *PipelineNumber* class stores all individual pipeline numbers. Each pipeline starts form some P&ID and ends to some other or the same P&ID which is the reason that the *Pipeline* class is associated with the *PID* class.

3.1.8 Motor List

The motor list was a document used by the electrical department to list all new or modified motors in the case project. The results of analyzing the most important columns of the motor list are shown in Table 10.

Table 10. The most important fields of the project's motor list from the instrumentation department's perspective.

Index	Data field name	Remarks
1	Motor tag	
2	Motor description	(service)
3	Scope	options: main project / turnaround
4	Nominal voltage (V)	options: 110/230/400/690/10000
5	Nominal current (A)	
6	Nominal power (kW)	
7	Feeder panel tag	
8	Starting method	options: DOL/VFD/SS/2-S
9	Field control switch type	options: push button / off-auto / hand-off-auto / N/A
10	Field control switch tag	
11	Hazardous area classification	zone 2 / non-hazardous area
12	P&ID	(drawing number)
13	Schematic diagram	(drawing number)
14	Battery limit	ISBL/OSBL
15	Related process unit or area	
16	Revision	revision for each motor
17	Document revision history	(many fields)

In addition to the fields listed in Table 10, instrumentation department needed information also about the analog signal ranges such as motor current measurement area, motor speed control area, and motor actual speed measurement area. Every motor is related to some process equipment that is shown in process equipment list. Equipment description field in the process equipment list equals the motor description/service field in the motor list. Also, most motors have some I/O connections to the control system and these are in particular interest of the instrumentation design engineer.

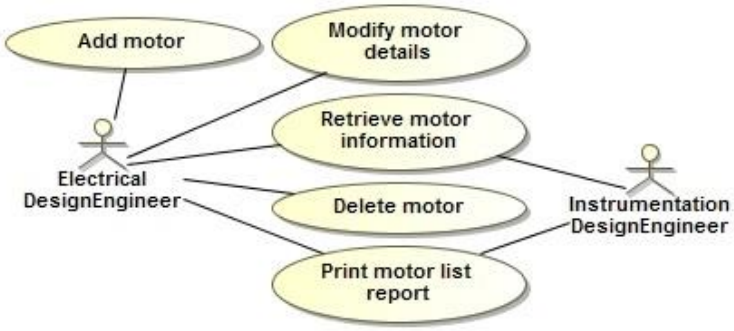


Figure 19. *The initial use case diagram for the motor list data.*

The initial use case diagram for the motor list data is shown in Figure 19. The electrical design engineer is responsible or executing all input use cases introduces in Figure 19 and the instrumentation design engineer is only supposed to retrieve motor information and print the motor list report. When we analyze the use cases in Figure 19, the required data in Table 10, and additional required field mentioned in this section, we

can create the initial class diagram for the motor list data. This class diagram is shown in Figure 20.

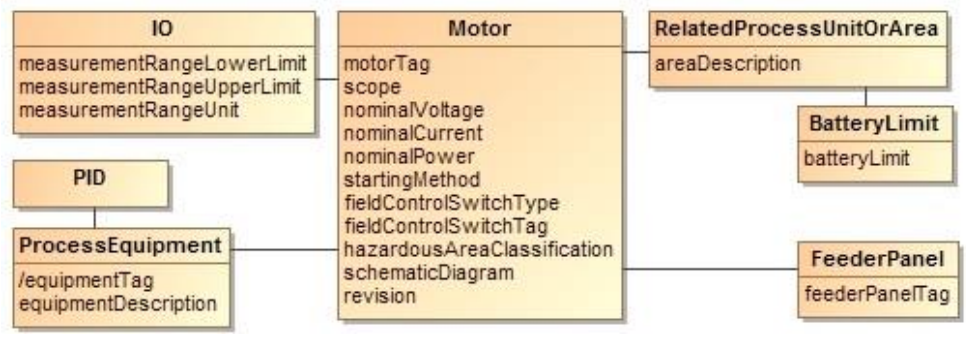


Figure 20. *The initial conceptual schema for the motor list data.*

The class diagram consists of seven classes. Class *I/O* includes all I/Os for all control systems of the project. Measurement range objects for the current and speed signals of the motors are located in this class. Feeder panels are objects as their own so they are modeled to have a dedicated class.

3.1.9 User Interface Requirements

The user interface of the prototype application should be as simple as possible. The main objective is to design an interface that can be used for testing that the conceptual schema is designed well. When the application is started, the first window appearing to the screen should be a main menu. The main menu should look as in Figure 21 having buttons for a user to navigate further in the application.

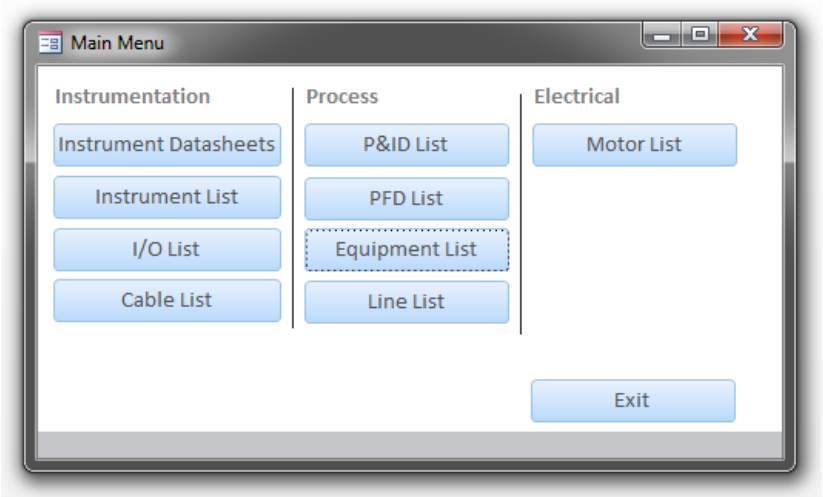


Figure 21. *The initial requirements for the main menu of the application.*

In Figure 21, the main menu is divided into three sections: instrumentation, process, and electrical. In addition to these, there is a button in the right down corner for the user to leave the application.

3.2 Conceptual Database Design

In the conceptual design phase, two models were created: the use case diagrams and class diagrams. The design of the models was started from improving the initial models created in the requirements definition phase. Both the use case diagrams and class diagrams were developed concurrently. During the development of the models, the requirements partially changed from the initial design as the models got their final form. For example, the model described in Figure 12 was simplified by removing the specialization level of two classes: *Sensor* and *Valve*. This modification is also in accordance with Lipták's example about relational model for instrumentation data which indicates that each instrument type should have a dedicated class and all these dedicated classes are then directly related to instrument class [16, pp. 19].

Both the use case diagrams and class diagrams were created by using GenMyModel which is a web based modeling platform. GenMyModel is run from a cloud server and it supports among others UML models. GenMyModel can be found in `www.genmymodel.com` and it is design for software architects, developers and business process analysts. [14] The use case diagrams developed for this thesis are shown in Appendix B and written descriptions for the diagrams are shown in Appendix C. The class diagrams are shown in Appendix D.

3.3 Logical Database Design

In the logical database design phase, the class diagrams developed in the previous design phase were mapped into relational model. The mapping was done by following the five-step mapping algorithm introduced in Section 2.5.3. The logical database design was carried out by using Microsoft Access. As a result, the relational model was created into Access meaning that all tables and relationships were readily made for the implementation phase. Access enables printing out a report of the relational model, so this function was used for the documentation of the relational model in this thesis. The report representing the relational model developed for the database application prototype is shown in Appendix E.

The logical database design phase started from mapping all regular classes into relations which are tables in Access. All attributes of the UML classes became fields of the tables in Access. For each table, one of the fields was selected as a primary key. The next phase was to map all binary 1 to 1 associations. In the class diagram model, there was only one binary 1 to 1 association: the association between classes *InstrumentAirManifoldOutput and AutomaticValve* which had been mapped into relations *tblInstrumentAirManifoldOutput* and *tblAutomaticValve* in the relational model. A relationship was created between the two relations by following a foreign key approach.

Access provides a graphical user interface where these relationships can be easily created by drawing a line between the relations that participate to the relationship. When the line is drawn, Access prompts a window providing the user to set three settings: should Access enforce referential integrity, should Access cascade updates, and should Access cascade deletions. For the relationships, the referential integrity should be enforced because otherwise the DBMS does not take care of the integrity rules between foreign keys and the primary keys. When a relationship line is drawn between two relations, Access automatically recognizes the relationship type to be 1 to N if no constraints are set. However, if a foreign key has been set to be a unique value, Access recognizes the relationship type to be 1 to 1 because a unique foreign key can point only to a unique primary key. In addition to the foreign keys in 1 to 1 relationships, there were also some other fields that required a unique value constraint. These fields were natural keys which were not primary keys because there was a surrogate key as a primary key in the relation. One example of this is the field *motorTag* in the table *Motor*. The table has a surrogate key field *ID* as a primary key and *motorTag* is a natural key that must have a unique value because two motors cannot have a same tag. One more constraint is related to cardinalities: If cardinality of the relationship is 1, the corresponding field in the relation table must be set to have a compulsory value in Access.

Almost all of the remaining associations were binary 1 to N associations. These were mapped by following the most common mapping approach where the relationships were created between the foreign key and the primary key of the relations. In some cases, there was a need to create more than one relationship between two relations. One example of this was two relationships between relations *tblPipelineNominalDiameter* and *tblPipeline:* one relationship for minimum and one for maximum diameter of the pipeline. Access does not allow drawing two relationship lines between the same relations but there is another way to represent these relationships. In Access, it is possible to create a copy of the relation participating to the 1-side of the 1 to N relationship and draw a relationship line to both of these tables from the relation participating to the N-side of the relationship. For example, relation *tblPipelineNominalDiameter* was copied and two relationship lines were drawn from *tblPipeline* to *tblPipelineNominalDiameter*.

The next phase was to map all binary M to N associations. There was only one association of this type in the class diagram model: the association between relations *Io* and *Interlock* which had been mapped into relations *tblIo* and *tblInterlock* in the relational model. This M to N association was mapped by creating a third table *tblIoInterlock* which included as foreign keys the primary key of the relations *tblIo* and *tblInterlock*. The fifth and final mapping step was to map all generalizations (or specializations). There were two cases where generalization had been used in the class diagram model. The first case was a superclass *Cable* which had four subclasses: *InstrumentFieldCable*, *InstrumentTrunkCable*, *GeneralCable*, and *MotorCable*. This generalization was mapped by using the third mapping technique for generalization/specialization

introduced in Section 2.5.3. The second case was a multilevel generalization. Superclass *AutomationTag* had subclasses *Instrument* and *Io*. Further, subclass *Instrument* was a superclass which had subclasses *AutomaticValve*, *PressureGauge*, *PressureTransmitter*, *FlowmeterMagnetic*, *TemperatureGauge*, and *TemperatureTransmitter*. These all were mapped by using the first mapping technique for generalization/specialization introduced in Section 2.5.3.

After mapping, normalization was applied to the whole relational model. In a consequence of well-done conceptual design phase, all relations were confirmed to be already in third normal form, so no modification of the model was needed.

3.4 Implementation

When the logical database design phase was completed, the next phase was to implement the database with Access. Resulting from the logical database design phase, the relational model was completely created into Access. This comprehended all tables, their fields and the relationships between the tables, which created the conceptual level of the three-schema architecture. The implementation phase comprehended creating the external level of the three-schema architecture, that is, the user interface. The user interface design followed the use case diagrams and descriptions where the interface between the end users and the application had been defined. The prototype application was given name *FLUXUS*.

First, main menu for the application was created by making an Access form with buttons leading to other forms of the application as defined in the requirements definition phase in Section 3.1.9. For all buttons, Access macros were used to create the actions that each button triggered. The result of the implementation is shown in Figure 22. In the bottom of the form, the main menu had three buttons: *Exit*, *About* and *Settings*. By pressing the *Exit* button, the user can leave and close the application. When pressing About, the application pops up a small window including some information about the application, such as the name, version, creator, and short description. When pressing *Settings*, the application pops up a small window containing options for setting the visibility attribute for a list of objects that are marked as *deleted* in the database application. For example, if some of the P&IDs are marked as *deleted* in the application, they are not shown in the P&ID list unless option *Show deleted P&IDs* is marked as *true* in the Settings window.

Figure 22. *The main menu of the implemented application prototype.*

After the main menu had been completed, it was turn for the other forms to be created. Each data form that were able be opened from the main menu were formed from tabs. For example, the *Automation Tag List* form shown in Figure 23 contained four tabs: *Automation Tag*, *Tag Type*, *Process Area*, and *Tag Number*.

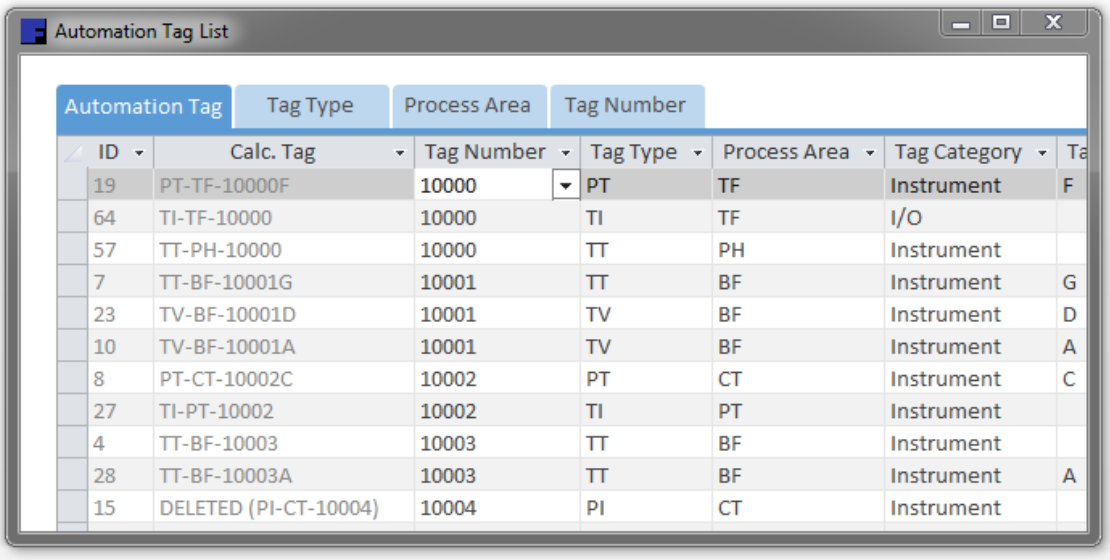


Figure 23. *Implemented Automation Tag List which was formed of four tabs: Automation Tag, Tag Type, Process Area, and Tag Number.*

Each tab contained a sub form of which almost all were made to look like tables where the user is able to see multiple records at once. The only exception was the instrument

datasheet window where the user is able to see only one record at once because each record was set to be on a separate page on the form.

Each sub form was accessing data on the Access tables. These sub forms were based on SQL queries. Queries defined what fields from the table were chosen, in which order the data was organized on the form, and also if the appearing of the data should depend on the options set in the Settings window. If there were calculated fields needed in the form whose value depends on values of the fields on the other tables, then this was defined also in the query. Below, there is an example of the SQL query on which the data on the sub form *I/O List* was based.

```
SELECT tblIo.ID, qryAutomationTagList.CalculatedAutomationTag,
tblIo.InstrumentID, tblIo.UnitOfMeasureID, tblIo.ControlSystemID,
tblIo.MotorID, tblIo.IoType, tblIo.SignalType, tblIo.SignalVoltage,
tblIo.AvailableMeasurementRangeLowerLimit,
tblIo.AvailableMeasurementRangeUpperLimit,
tblIo.SetMeasurementRangeLowerLimit, tblIo.SetMeasurementRangeUpperLimit,
tblIo.AlarmTripValue, tblIo.Location, tblIo.ControllerType, tblIo.Remarks,
tblIo.InternalNotes, tblIo.Revision, qryAutomationTagList.Deleted

FROM tblIo INNER JOIN qryAutomationTagList ON tblIo.ID = qryAutomation-
TagList.ID

WHERE (((qryAutomationTagList.Deleted)=False Or (qryAutomation-
TagList.Deleted)=GetGlobal("gboolShowDeletedIo")) AND ((qryAutomation-
TagList.TagCategory) Like "I/O"))

ORDER BY qryAutomationTagList.CalculatedAutomationTag;
```

In the *WHERE* section of the SQL query, there is a global variable *gboolShowDeletedIo* whose value the user can change in the Settings widow which can be accessed from the main menu of the application. The global variables were defined in a VBA Module in the Access application.

Instrument and I/O objects were designed to have the same ID number as primary key through the whole hierarchy of the generalization model. For example, an object in the table *tblTemperatureTransmitter* has the same ID number as a primary key as the related objet in the tables *tblInstrument* and *tblAutomationTag*. When a new automation tag object is created, Access automatically creates an ID number for it because the data type of its primary key was set to be *auto number*. However, Access is not able to create the same ID number for other objects in the generalization hierarchy unless VBA code is utilized. A special buttons *Add Instrument* and *Add I/O* were created on forms *Instrument Datasheet* and *I/O List* respectively. When a user presses these buttons, they run a VBA code which creates a new object (instrument or I/O) which has the same ID number as primary key as the automation tag for which the user is creating the new object.

One button on the main menu of the application was *Reports* button. When a user clicks this button, a new pop-up window opens giving the user a possibility to choose which report he would like to create. For each report type, a unique button was created on a pop-up window but because the application is only a prototype, only two of the buttons were set as enabled. These buttons were for *I/O List* and *Temperature Transmitter datasheet* reports. If the user clicks for example the *Temperature Transmitter datasheet* report button, the application creates this report and opens it in a print preview. The user can then either just view the report or also print it.

The implementation of the database application comprehended the following amount of Access objects: 39 tables, 40 queries, 61 forms, 2 reports, and 1 code module. Six of the forms included VBA code running behind the form. The total amount of VBA code in the application was 330 rows including also empty rows and comments. RVBA Naming Conventions were used throughout the implementation in naming all Access objects, such as tables, fields, queries, forms, buttons, reports, VBA modules, and variables.

4. VALIDATION AND TESTING

Testing of the application consisted of two parts. First, testing that the application was implemented as designed and did not include major errors. Second, validating that the research problem, decreasing data redundancy and separating data from report files, was solved for a set of case project's data.

4.1 Ad-Hoc and Model-Based Testing

When the application was implemented, it was tested by using so called Ad-hoc testing. Ad-hoc testing is the most common testing method and it means that the application is tested by checking that everything seems to work as it should. In ad-hoc testing, the testing process is not usually documented anyhow which was the case also for this thesis.

In addition to ad hoc testing, the application was tested using model-based testing where the application tests were based on models, in this case UML use case diagrams. In model-based testing, the use case models were used to derive test cases to test the application. The objective for the model-based testing is to see if the application fulfills its requirements. [13, pp. 74–75, 213] Test case is a set of test case preconditions, inputs, and expected results, developed to drive the execution of a test item to meet test objectives. Test Case Specification is a documentation of a set of one or more test cases. [20, pp. 7] Test Case template used in this thesis was derived both from IEEE standard and an example template by Dr Ghazy Assassa. [21, pp. 102] [22]

Test cases should have a unique identifier so that each test case can be distinguished from all other test cases. [21, pp. 31] In the test cases of this thesis, the identifier is called *Test Case ID* and it follows a syntax which is similar to the one used use case IDs. Preconditions describe the required state of the test environment and any special constraints pertaining to the execution of the test case. Inputs specify each action required to bring the test item into a state where the expected result can be compared to the actual results. Expected system response specifies the expected outputs and behavior required of the test item in response to the inputs that are given to the test item when it is in its precondition state. [21, pp. 31–32] Actual system response is then compared to the expected system response to determine whether the test case passes or fails the test.

The test case specification of this thesis is shown in Appendix F. As a result of executing the test cases, 98,6 % of the tests passed successfully and 1,4 % of the tests failed. High value for passed tests indicates that the Ad-hoc testing was done thoroughly

during the implementation phase. The reason for all failures was that there were some settings in Microsoft Access that were forgotten to be configured correctly during the implementation phase. In this thesis, the minor errors representing 1,4 % of the tests were not corrected in the application because it was developed only for prototyping purposes. Passing 98,6 % percent of the tests was enough to prove that the application was successfully implemented as designed.

4.2 Validation of the Solution for the Research Problem

The research problem was to develop a relational database application prototype in order to separate instrumentation engineering data from report files and reduce data redundancy compared to the file-system approach. In this section, it is shown with examples how much better the relational database application prototype is compared to the file-system approach used in the case project. This included measuring how much data redundancy decreased for a certain set of data and how much time could be saved in a certain situation when data is separated from report files.

Because the amount of data in the case project was very massive, the data redundancy check for the whole case project's data would be too arduous. For this reason, the check was performed only for a set of the project's data. This set was chosen to be ISBL instrument list data including only the following six instrument types: automatic valves, magnetic flowmeters, pressure and temperature transmitters, and pressure and temperature gauges. On this instrument list, there were 592 rows for equal amount of instruments and 22 columns for equal amount of details for these instruments, such as manufacturer, model, and P&ID. The check was performed so that first the amount of atomic data strings on case project's file-based instrument list was calculated. The atomic data string means a piece of information contained in a single cell of the instrument list and it is equivalent to a single attribute value of a database relation.

In addition to calculating the amount of atomic data strings on the instrument list, it was also calculated how big amount of this data had unique value and how big amount had redundant value. For example, on column *P&ID*, there were 74 different P&ID document numbers meaning that 74 instruments had a unique P&ID document number value and the rest 518 instruments had a P&ID document number that was redundant to those 74 unique values. The result of the calculations for each instrument list column is shown in Figure 24 and the result of the calculation for the instrument list as a whole is shown in Figure 25.

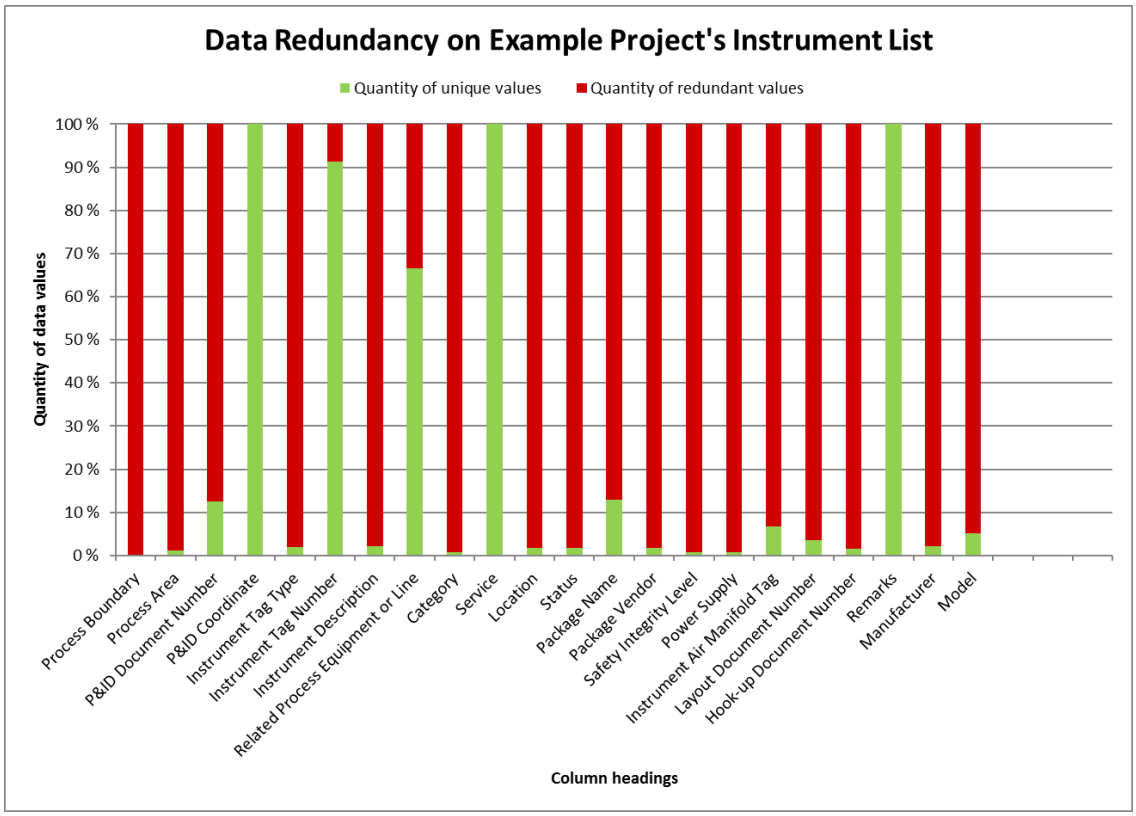


Figure 24. *The amount of unique and redundant data values on the case project's instrument list containing only automatic valves, magnetic flowmeters, pressure and temperature transmitters, and pressure and temperature gauges.*

In Figure 24, the amount of data values is represented as percentages where 100 % equals 592 data values. It is clear from Figure 24 that most of the data on the instrument list was redundant and Figure 25 shows that actually 77 % of all data on the instrument list was redundant. If the same information represented on the instrument list was saved on the database application prototype developed in this thesis, only the unique values representing 23 % of the total amount of data needed to be saved on the database because this data would be normalized into third normal form. This means that the data saved into the database would be consistent, it would be easier to be updated, and likelihood for errors would decrease. For example, P&ID document number *PT-9T-057* was repeated on case project's instrument for 35 instruments. This means that if the P&ID document number must be corrected from *PT-9T-057* into, for example, *PT-9T-57*, then this update must be made for all 35 instruments. This takes more time and there is a risk that, by mistake, the update is made only for part of the 35 instruments leading to situation that the same P&ID documents number is being referred inconsistently with two different naming conventions.

In database application prototype instead, P&ID document number *PT-9T-057* is saved only once to the P&ID list and all 35 instruments are referring to the primary key of this P&ID with foreign keys. Then if the P&ID document number must be corrected, the change update needs to be done only once and because the primary key of the P&ID

document number does not change, the 35 instruments will automatically refer to the corrected P&ID document number consistently.

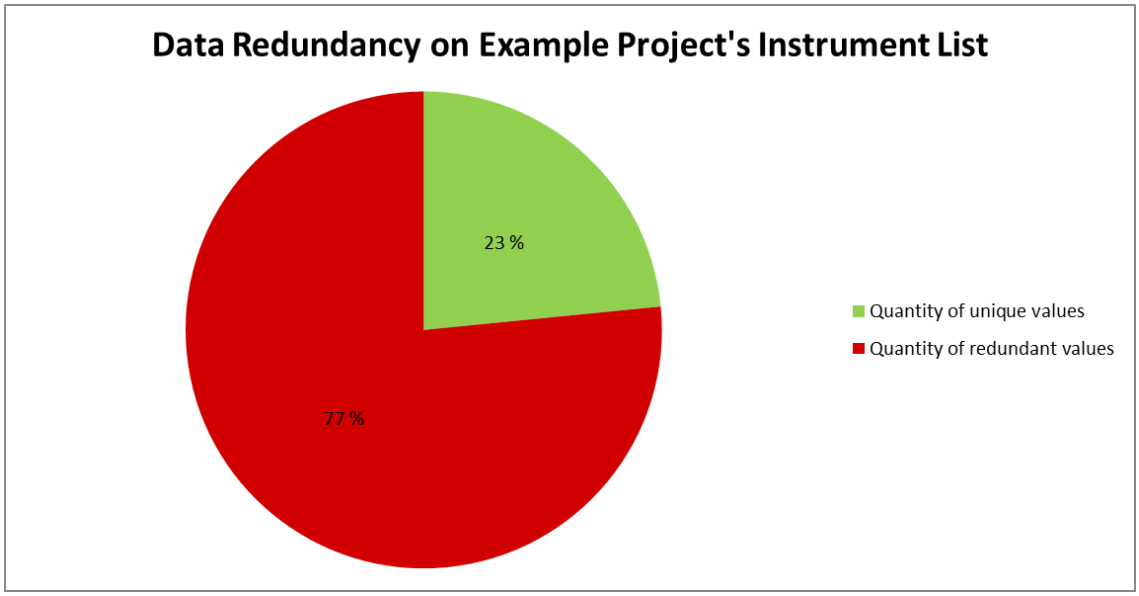


Figure 25. *The total amount of unique and redundant data values on case project's instrument list containing only automatic valves, magnetic flowmeters, pressure and temperature transmitters, and pressure and temperature gauges.*

Figures 24 and 25 are examples of only small part data redundancy in case project's file-based information control. Redundant data within the same document is still somewhat easy to keep up-to-date but more difficult is to handle data-redundancy between separate documents. For example, if instrument tag number changed for some instrument of the case project, this would mean that the same update needed to be made at least on instrument list, I/O list, instrument datasheet, and cable list. In database application prototype instead, it would be enough to update the number only once on automation tag list.

In addition to calculating an example of how much data redundancy could be decreased, it was also estimated how much time could be saved in a situation where datasheet templates should be changed for all instrument types for which the datasheet was created in the database application prototype compared to the time that would be required when using file-based approach. This is important estimation because there was actually a case in the case project that the client required changing the datasheet template for all 800 instruments. In file-based approach this meant that the data included into all 800 instrument datasheet needed to be transferred to new templates. Even though macro programming was used to execute work faster, it took still tremendous amount of time compared to the time that would have been required if database approach had been used.

In file-based approach, data is located in the same file with the datasheet template and thus each datasheet must be modified individually. In database approach, data and datasheet templates are separated and thus it is enough to modify only each datasheet report template and then populate the reports with the data from the database. For example, if there were hundred temperature transmitters using the same datasheet template, in datasheet approach, it is enough to change only one template but in file-based approach, it is required to do the same changes to all hundred datasheets.

In the database application prototype, there were instrument datasheet templates implemented for six instrument types: automatic valve, temperature transmitter, temperature gauge, pressure transmitter, pressure gauge, and magnetic flowmeter. In the case project the amount of these instrument types was 592 pieces in total. If it takes one hour of work to create a new template for one instrument type and five minutes of work to move data from an old template to the ne template in file-based approach, we can estimate the time that can be saved when doing the same changes using a database approach. In file-based approach the time required would be six hours for changing the templates and 49 h for populating the templates with the data from the old templates. This makes 55 hours in total. With database approach instead, only six hours would be needed to change the database templates and populating the reports with the data happens automatically taking virtually no time. Thus, in this example, the same amount of work could be done 89 % faster with database approach than with file-based approach. The amount of work hours needed for the tasks are only rough estimations based on real-life experience from the case project but it gives good image about the vicinity of time that can be saved in some situations when using database approach instead of file-based approach.

Reducing data redundancy and separating data from report templates saves time and improves data consistency but these are just some examples of the benefits of the database approach over the file-based approach. The measured results for these were the easiest to estimate but there are also other benefits of database approach that are more difficult to measure. These benefits are at least support for multiple views and multiple users, having the data structure and constraints defined within the database itself, and possibility to create queries and reports fast from the whole data content. However, all these benefits come with the cost that a database application takes more time to design and implement than simple files but if there is a good starting point for the database development, the time can be remarkably decreased. That was also the motivation for this thesis.

5. CONCLUSION

The research problem of this thesis was to study how to dispose of the disadvantages of the file-system approach in information control of detailed instrumentation engineering data. The main objectives of the solution were to reduce data redundancy and separate data from reports. The problem was solved for one case project's data by developing a relational database application prototype designed to hold detailed instrumentation data of the case project. The objective was also to find and document a systematic development process and data models that can be used in future projects for developing a new project-specific database application for information control.

The development process started from defining the initial requirements for the database application by analyzing case project's reports, such us, instrument list and instrument datasheets. Then, the initial requirements were used to develop two models: use cases and conceptual schema. These models were created by using UML use case and class diagrams. After this, a five-step mapping algorithm was used to transfer the conceptual model into relational model. The database application was implemented with Access, a desktop database application from Microsoft. The relational model was implemented into Access by defining the tables, fields, keys, and relationships between the tables. Use case diagrams were used to implement the user interface by creating the Access form and reports. Finally, the application was tested by using ad-hoc and model-based testing. As a result, 98,6 % of tests cases were passed successfully which indicated well-done design.

The database application prototype had all data of the case project in third normal form meaning that data redundancy was minimized. Each entity of the miniworld got its own relation holding the attributes of the entity. Because of the massive amount of case project's data, it was not practical to calculate the amount of redundant data in all case project's reports. Instead, the calculation was performed for a set of data on case project's instrument list. The calculation revealed that 77 % of the data on the data set was redundant. If this same data was saved on the database application developed in this thesis, this redundancy would be avoided. In addition to checking data redundancy, it was also estimated how much time could be saved if datasheet layout needed to be changed for 592 instruments and the work was carried out by using the database application prototype instead of the filed-based application of the case project. The result of the estimation was that with database application prototype it would be possible to carry out the work 89 % faster. This results from the fact that in the database application prototype the data is separated from the external view of the application which enables report layout modifications without affecting the data anyhow.

In addition to reducing data redundancy and separating data from the external view, the database application prototype also enable even multiple engineering disciplines working simultaneously with the same application and data set. This improves communication, makes data analysis more efficient and reduces likelihood for errors. For the database application prototype, it is also possible to create more views, either forms or reports, for different user groups, for example, for different engineering disciplines. The application can be further developed to include also for example user authentication, automatic revision control, or wiring diagram design. Many monotonous tasks can be automated by using macros, VBA code or action queries. Import functions of Access can be used to bring data to the database from for example spreadsheet applications and with export functions the data can be again transferred out from the database. The database can be also expanded to Microsoft SQL Server or Share Point. The application can be developed to be able to handle data for multiple projects and multiple clients. User interface can be further developed, for example, by utilizing theory about user-friendly user interfaces and by making usability tests with users.

The documentation of the development process and ready data models introduced in this thesis offer a ready starting point for developing new database applications for engineering projects that do not have any other database application readily available. However, the time needed for successful design of the application might limit the possibility to use the database approach in many projects.

REFERENCES

[1] Ramez Elmasri and Shamkant B. Navathe. Fundementals of Database Systems. Fifth Edition. 2007. Addison-Wesley. 1123 p.

[2] Michael Alexander and Dick Kusleika. Access 2013 Bible. First Edition. Indianapolis, Indiana. 2013. John Wiley & Sons. 1250 p.

[3] Simon Bennett, Steve McRobb, and Ray Farmer. Object-Oriented Systems Analysis and Design Using UML. Fourth Edition. 2010. UK. McGraw-Hill. 714 p.

[4] Tampere University of Technology. 2015. TIE-22100 Johdatus tietokantoihin [WWW]. [Referenced 12.7.2015] Available: https://idle.cs.tut.fi/idle/course/showcoursefile.php?cid=137&coid=1&file=Luento1.2015.pdf.

[5] Tampere University of Technology. 2015. TIE-22100 Johdatus tietokantoihin [WWW]. [Referenced 12.7.2015] Available: https://idle.cs.tut.fi/idle/exercise/task.php?cid=137&eid=3071&tid=502.

[6] Information technology. Database languages. SQL. Part 1: Framework (SQL/Framework). ISO/IEC 9075-1. 2011.

[7] Simon Bennett, John Skelton and Ken Lunn. Schaum's Outline of UML. Second Edition. 2005. UK. McGraw-Hill. 398 p.

[8] Clare Churcher. Beginning Database Design. 2007. United States of America. Apress. 240 p.

[9] Tampere University of Technology. 2015. TIE-22200 Tietokantojen Suunnittelu [WWW]. [Referenced 5.9.2015] Available: https://idle.cs.tut.fi/idle/course/showcoursefile.php?cid=135&coid=14&file=Luento1.2014-2015.pdf.

[10] International Institute of Business Analysis. 2015 [WWW]. [Referenced 17.9.2015] Available: austin.iiba.org/download/UseCase.dot

[11] Alexander Kossiakoff, William N. Sweet, Sam Seymour, and Steven M. Biemer. Systems Engineering Principles and Practice. Second Edition. 2011. Hoboken, New Jersey. John Wiley & Sons, Inc. 560 p.

[12] Greg Reddick. 2009. RVBA Conventions [WWW]. [Referenced 19.12.2015] Available: http://www.xoc.net/standards/

[13] Jussi Pekka Kasurinen. Ohjelmistotestauksen käsikirja. First Edition. 2013. Saarijärvi. Saarijärven Offset Oy. 236 p.

[14] GenMyModel. 2016. GenMyModel [WWW]. [Referenced 19.3.2016] Available: https://www.genmymodel.com/

[15] Kirill Fakhroutdinov. 2016. The Unified Modeling Language [WWW]. [Referenced 3.4.2016] Available: http://www.uml-diagrams.org/use-case-include.html

[16] Béla G. Lipták. Instrument Engineers' Handbook. Process Software and Digital Networks. Third Edition. 2002. Boca Raton, Florida, USA. CRC Press. 912 p.

[17] Crystal Long. 2008. Access Basics for Programming: Types of Applications [WWW]. [Referenced 9.4.2016] Available: http://www.allenbrowne.com/binary/Access_Basics_Crystal_080220_Chapter_01.pdf

[18] Crystal Long. 2008. Access Basics for Programming: Database Objects [WWW]. [Referenced 9.4.2016] Available: http://www.allenbrowne.com/binary/Access_Basics_Crystal_080220_Chapter_02.pdf

[19] Microsoft Corporation. Kraig Brockschmidt and Lorenz Prem. 2010. Microsoft Data Development Technologies: Past, Present, and Future [WWW]. [Referenced: 9.4.2016] Available: https://msdn.microsoft.com/en-us/library/ee730343.aspx

[20] Software and systems engineering. Software testing. Part 1: Concepts and definitions. ISO/IEC/IEEE 29119-1. 2013.

[21] Software and systems engineering. Software testing. Part 3: Test Documentation. ISO/IEC/IEEE 29119-3. 2013.

[22] Dr Ghazy Assassa. 2004. Software Engineering. Test Case Template and Examples [WWW]. [Referenced: 10.4.2016] Available: http://bu.edu.eg/portal/uploads/Engineering,%20Shoubra/Mechanical%20Engineering/589/crs-11721/Files/Test%20Case%20Template.pdf

APPENDIX A

Use Case List

Use Case ID	Primary Actor	Use Cases

1 Feature Name (Example: ATM Transaction)

1.1 Feature Process Flow / Use Case Model

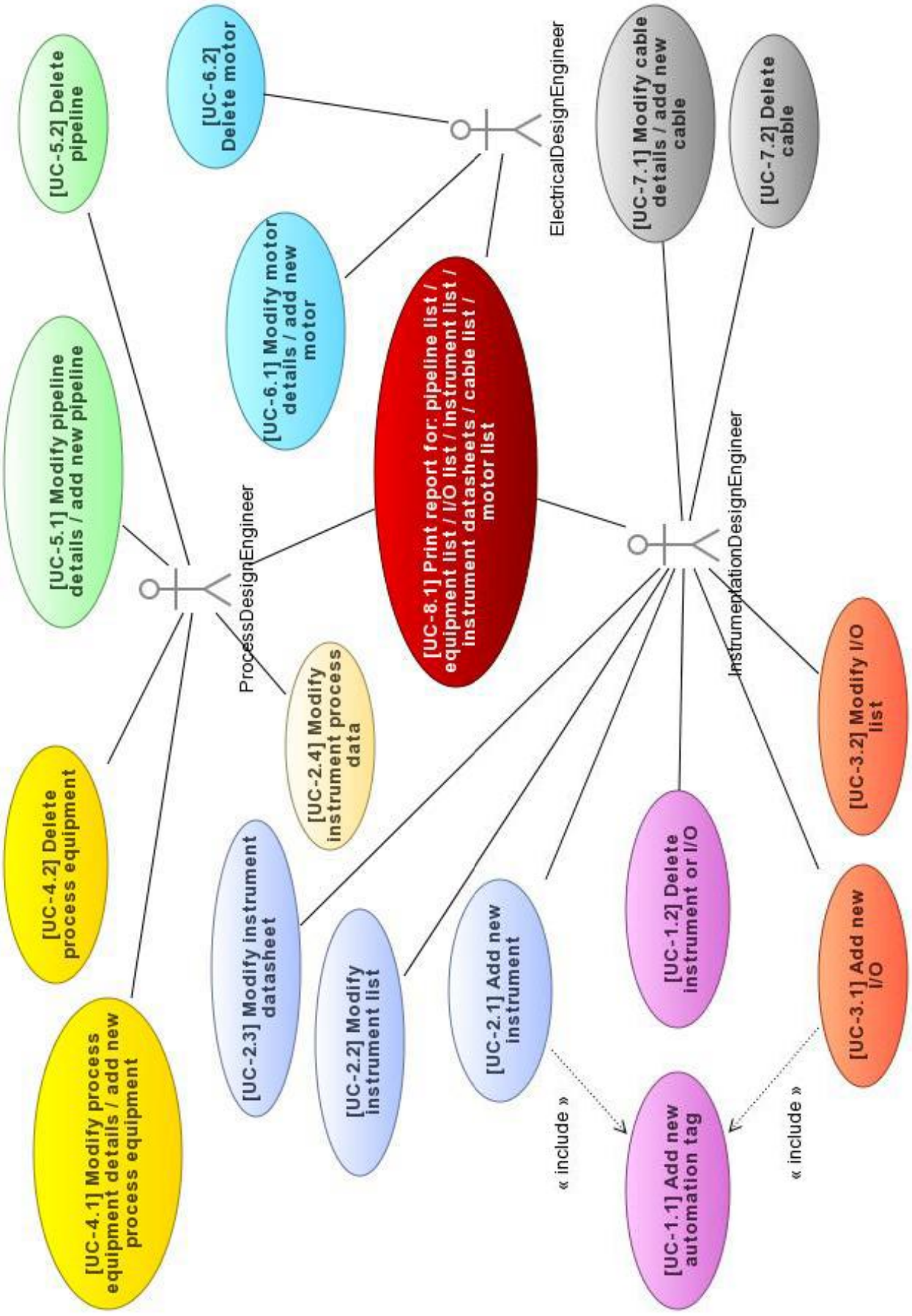
1.2 Use Case(s)

Use Case ID:	Enter a unique numeric identifier for the Use Case. e.g. UC-1.2.1		
Use Case Name:	Enter a short name for the Use Case using an active verb phrase. e.g. Withdraw Cash		
Created By:		**Last Updated By:**	
Date Created:		**Last Revision Date:**	
Actors:	[An actor is a person or other entity external to the software system being specified who interacts with the system and performs use cases to accomplish tasks. Different actors often correspond to different user classes, or roles, identified from the customer community that will use the product. Name the actor that will be initiating this use case (primary) and any other actors who will participate in completing the use case (secondary).]		
Description:	[Provide a brief description of the reason for and outcome of this use case.]		
Trigger:	[Identify the event that initiates the use case. This could be an external business event or system event that causes the use case to begin, or it could be the first step in the normal flow.]		
Preconditions:	[List any activities that must take place, or any conditions that must be true, before the use case can be started. Number each pre-condition. e.g. 1. Customer has active deposit account with ATM privileges 2. Customer has an activated ATM card.]		
Postconditions:	[Describe the state of the system at the conclusion of the use case execution. Should include both *minimal guarantees* (what must happen even if the actor's goal is not achieved) and the *success guarantees* (what happens when the actor's goal is achieved. Number each post-condition. e.g. 1. Customer receives cash 2. Customer account balance is reduced by the amount of the withdrawal and transaction fees]		
Normal Flow:	[Provide a detailed description of the user actions and system responses that will take place during execution of the use case under **normal, expected** conditions. This dialog sequence will ultimately lead to accomplishing the goal stated in the use case name and description. 1. Customer inserts ATM card 2. Customer enters PIN 3. System prompts customer to enter language performance English or Spanish 4. System validates if customer is in the bank network 5. System prompts user to select transaction type		

APPENDIX A

	6. Customer selects Withdrawal From Checking 7. System prompts user to enter withdrawal amount 8. … 9. System ejects ATM card]
Alternative Flows: **[Alternative Flow 1 – Not in Network]**	[Document **legitimate** branches from the main flow to handle special conditions (also known as extensions). For each alternative flow reference the branching step number of the normal flow and the condition which must be true in order for this extension to be executed. e.g. Alternative flows in the *Withdraw Cash* transaction: 4a. In step 4 of the normal flow, if the customer is not in the bank network 1. System will prompt customer to accept network fee 2. Customer accepts 3. Use Case resumes on step 5 4b. In step 4 of the normal flow, if the customer is not in the bank network 1. System will prompt customer to accept network fee 2. Customer declines 3. Transaction is terminated 4. Use Case resumes on step 9 of normal flow Note: Insert a new row for each distinctive alternative flow.]
Exceptions:	[Describe any anticipated **error conditions** that could occur during execution of the use case, and define how the system is to respond to those conditions. e.g. Exceptions to the Withdraw Case transaction 2a. In step 2 of the normal flow, if the customer enters and invalid PIN 1. Transaction is disapproved 2. Message to customer to re-enter PIN 3. Customer enters correct PIN 4. Use Case resumes on step 3 of normal flow]
Includes:	[List any other use cases that are included (“called”) by this use case. Common functionality that appears in multiple use cases can be split out into a separate use case that is included by the ones that need that common functionality. e.g. steps 1-4 in the normal flow would be required for all types of ATM transactions- a Use Case could be written for these steps and “included” in all ATM Use Cases.]
Frequency of Use:	[How often will this Use Case be executed. This information is primarily useful for designers. e.g. enter values such as 50 per hour, 200 per day, once a week, once a year, on demand etc.]
Special Requirements:	[Identify any additional requirements, such as nonfunctional requirements, for the use case that may need to be addressed during design or implementation. These may include performance requirements or other quality attributes.]
Assumptions:	[List any assumptions that were made in the analysis that led to accepting this use case into the product description and writing the use case description. e.g. For the *Withdraw Cash* Use Case, an assumption could be: The Bank Customer understands either English or Spanish language.]
Notes and Issues:	[List any additional comments about this use case or any remaining open issues or TBDs (To Be Determined) that must be resolved. e.g. 1. What is the maximum size of the PIN that a use can have?]

APPENDIX B


[UC-5.2] Delete pipeline
[UC-6.2] Delete motor
[UC-7.1] Modify cable details / add new cable
[UC-7.2] Delete cable
ElectricalDesignEngineer
[UC-5.1] Modify pipeline details / add new pipeline
[UC-6.1] Modify motor details / add new motor
[UC-8.1] Print report for: pipeline list / equipment list / I/O list / instrument list / instrument datasheets / cable list / motor list
ProcessDesignEngineer
InstrumentationDesignEngineer
[UC-4.2] Delete process equipment
[UC-2.4] Modify instrument process data
[UC-3.2] Modify I/O list
[UC-2.3] Modify instrument datasheet
[UC-2.2] Modify instrument list
[UC-2.1] Add new instrument
[UC-1.2] Delete instrument or I/O
[UC-4.1] Modify process equipment details / add new process equipment
« include »
[UC-1.1] Add new automation tag
[UC-3.1] Add new I/O
« include »

1. Automation Tag

Use Case ID:	UC-1.1		
Use Case Name:	Add new automation tag		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	19.11.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: instrumentation design engineer (IDE)		
Description:	IDE adds a new automation tag to the database either for a new instrument or a new I/O.		
Trigger:	IDE notices that there is an automation tag that must be added to the database.		
Preconditions:	1. The application is successfully started and the *Main menu* is open.		
Post conditions:	1. The new automation tag is added to the *Automation tag list*. 2. The new automation tag is available for IDE to choose if he is adding a new instrument or a new I/O to the database.		
Normal Flow:	1. IDE selects *Automation Tag List* on the *Main Menu* of the application. 2. *Automation Tag List* window opens. 3. IDE selects *Tag Type* tab. 4. IDE adds the tag type to the end of the list. 5. IDE selects *Process Area* tab. 6. IDE adds the process area to the end of the list. 7. IDE selects *Tag Number* tab. 8. IDE adds the tag number to the end of the list. 9. IDE selects *Automation Tag* tab. 10. IDE adds the new tag to the end of the list by selecting/typing the tag type, process area, tag number, tag suffix, scope, and P&ID. 11. IDE selects that *Tag category = Instrument* or that *Tag category = I/O* depending on whether the tag comes for an instrument or I/O. 12. IDE closes the *Automation Tag List* window. 13. The program returns to *Main Menu* window.		
Alternative Flows:	4a. In step 4 the of the normal flow, if IDE notices that the tag type already exists in the database before saving the new row. 1. IDE presses *Esc* key on keyboard. 2. Use Case resumes on step 5 of the normal flow. 6a. In step 6 the of the normal flow, if IDE notices that the process area already exists in the database before saving the new row. 1. IDE presses *Esc* key on keyboard. 2. Use Case resumes on step 7 of the normal flow. 8a. In step 8 the of normal flow, if IDE notices that the tag number already exists in the database before saving the new row. 1. IDE presses *Esc* key on keyboard. 2. Use Case resumes on step 9 of the normal flow. 10a. In step 10 of the normal flow, if the P&ID is not yet shown on the drop-down list from which the correct data should be selected. 1. IDE presses *Esc* key on keyboard. 2. IDE closes the *Automation Tag List* window. 3. IDE selects *Equipment List* on the *Main Menu* of the application. 4. *Process Equipment List* window opens. 5. IDE opens *P&ID* tab from the Process equipment list window. 6. IDE adds the new P&ID with all needed details to the end of the list. 7. IDE closes the *Process Equipment List* window. 8. Use Case resumes on step 1 of the normal flow.		

Exceptions:	4a. In step 4 the of the normal flow, if IDE does not notice that the tag type already exists in the database before trying to save the new row. 1. The application prompts an error message indicating that IDE has tried to add a duplicate value to the database. 2. IDE clicks *OK* and presses *Esc* key on keyboard. 3. Use Case resumes on step 5 of the normal flow. 6a. In step 6 the of the normal flow, if IDE does not notice that the process area already exists in the database before trying to save the new row. 1. The application prompts an error message indicating that IDE has tried to add a duplicate value to the database. 2. IDE clicks *OK* and presses *Esc* key on keyboard. 3. Use Case resumes on step 7 of the normal flow. 8a. In step 8 the of normal flow, if IDE does not notice that the tag number already exists in the database before trying to save the new row. 1. The application prompts an error message indicating that IDE has tried to add a duplicate value to the database. 2. IDE clicks *OK* and presses *Esc* key on keyboard. 3. Use Case resumes on step 9 of the normal flow.
Includes:	N/A
Frequency of Use:	3000 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	1. This Use Case is included by *Add instrument* and *Add I/O* Use Cases.

Use Case ID:	UC-1.2		
Use Case Name:	Delete instrument (or I/O)		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	19.11.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: instrumentation design engineer (IDE)		
Description:	IDE deletes an instrument (or I/O). After the deletion, the instrument will disappear from the instrument datasheets and instrument list (or the I/O will disappear from the I/O list).		
Trigger:	IDE notices that there is an instrument (or I/O) that must be deleted from the database.		
Preconditions:	1. The application is successfully started and the main menu is open.		
Post conditions:	1. The instrument disappears from the instrument datasheets, instrument list, and automation tag list (or the I/O disappears from the I/O list and automation tag list).		
Normal Flow:	1. IDE selects *Automation Tag List* from the *Main menu* of the application. 2. *Automation Tag List* window opens. 3. IDE selects *Automation Tag* tab. 4. IDE marks the instrument (or I/O) as *Deleted*. 5. The application prompts a warning message “You are about to mark the tag as deleted. Are you sure?”. 6. IDE clicks *OK*. 7. The application adds word “DELTED” in front of the tag that was just marked as *Deleted*. 8. IDE closes the *Automation Tag List* window. 9. The focus returns on the *Main Menu* of the application.		

Alternative Flows:	6a. In step 6 the of the normal flow, if IDE does not want to mark the tag as *Deleted.* *1.* IDE clicks *Cancel*. *2.* The warning window closes. *3.* Use Case resumes on step 4 of the normal flow.
Exceptions:	N/A
Includes:	N/A
Frequency of Use:	500 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	As a post condition, the instrument (or I/O) tag disappears from the automation tag list but it can be made visible again by checking a check box “Show deleted automation tags” in the *Settings* window.

2. Instrument

Use Case ID:	UC-2.1		
Use Case Name:	Add new instrument		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	19.11.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: instrumentation design engineer (IDE)		
Description:	IDE creates a new instrument to the database. After the addition, the instrument will appear to the instrument datasheets and instrument list.		
Trigger:	IDE notices that there is an instrument that must be added to the database.		
Preconditions:	1. The application is successfully started and the main menu is open. 2. The tag of the instrument has been successfully added according to Use Case UC-1.1 where in step 11 of the normal flow, IDE has chosen that *Tag category* = *Instrument*.		
Post conditions:	1. The new instrument appears to the instrument datasheets 2. The new instrument appears to the instrument list		
Normal Flow:	1. IDE selects *Datasheets* from the *Main Menu* of the application. 2. *Instrument Datasheets* window opens. 3. IDE clicks *Add New Instrument* button. 4. *Add New Instrument* window opens. 5. IDE selects the previously added automation tag and selects the correct instrument type. 6. IDE clicks *OK.* 7. *Add New Instrument* window closes and the focus returns to *Instrument Datasheets* window. The new instrument appears to instrument datasheets and instrument list.		
Alternative Flows:	5a. In step 5 of the normal flow, if IDE clicks *Cancel* instead of *OK*. 1. The transaction is terminated and changes are not saved. 2. *Add New Instrument* window closes and the focus returns to *Instrument Datasheets* window. 3. Use Case resumes on step 3 of the normal flow.		

Exceptions:	5a. In step 5 of the normal flow, if there are no tags available to choose for due to not added / incorrectly added automation tag. 1. IDE clicks *Cancel* instead of *OK*. 2. *Add New Instrument* window closes and the focus returns to *Instrument Datasheets* window. 3. IDE closes *Instrument Datasheets* window. 4. Use Case resumes on step 1 of the normal flow of Use Case UC-1.1. 5b. In step 5 of the normal flow, IDE clicks *OK* before the input of all needed information. 1. The application prompts an error message describing the first missing information starting from the top of the form. 2. IDE clicks *OK*. 3. Use Case resumes on step 5 of the normal flow. 5c. In step 1 of the exception 5a, IDE clicks *OK* instead of *Cancel*. 1. Use Case resumes on step 1 of the exception 5b.
Includes:	UC-1.1 (Add new automation tag)
Frequency of Use:	1000 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	N/A

Use Case ID:	UC-2.2		
Use Case Name:	Modify instrument list		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	4.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: instrumentation design engineer (IDE)		
Description:	IDE modifies detailed instrument data on the instrument list.		
Trigger:	IDE notices that there is some detailed instrument data that must be updated to the instrument list.		
Preconditions:	1. The application is successfully started and the main menu is open. 2. The instrument has been successfully added to the instrument list according to Use Case UC-2.1.		
Post conditions:	1. The updated data will appear on instrument list (and instrument datasheet).		
Normal Flow:	1. IDE selects *Instrument List* from the *Main Menu* of the application. 2. *Instrument List* window opens. 3. IDE selects *Instrument List* tab on *Instrument List* window. 4. IDE finds the correct row on the instrument list and updates the fields that are needed to be updated. 5. IDE closes *Instrument List* window. 6. *Instrument List* window closes and the focus returns to the *Main Menu* of the application.		
Alternative Flows:	4a. In step 4 of the normal flow, if IDE wants to add document number detail referring to instrument datasheet, hook-up, or location plan, but the document number is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE selects *Document List* tab from *Instrument List* window. 2. IDE adds the new document number with required details to the end of *Document List*.		

	3. Use Case resumes on step 3 of the normal flow. 4b. In step 4 of the normal flow, if IDE wants to add procurement code detail but the code is not yet shown in the drop-down list from which the correct data should be selected. 4. IDE selects *Procurement Codes* tab from *Instrument List* window. 5. IDE adds the new procurement code with required details to the end of *Procurement Codes list*. 6. Use Case resumes on step 3 of the normal flow. 4c. In step 4 of the normal flow, if IDE wants to add package unit name detail but the package unit name is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE selects *Package Units* tab from *Instrument List* window. 2. IDE adds the new package unit name with required details such as package vendor to the end of *Package Units* list. 3. Use Case resumes on step 3 of the normal flow. 4d. In step 2 of the alternative flow, if IDE wants to add package vendor detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE selects *Package Vendors* tab from *Instrument List* window. 2. IDE adds the new package vendor with required details to the end of *Package Vendors* list. 3. Use Case resumes on step 1 of the alternative flow 4c. 4e. In step 4 of the normal flow, if IDE wants to add a related process equipment detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. Use Case resumes on step 1 of the normal flow of Use Case UC-4.1. 2. Use Case returns from UC-4.1 and resumes on step 1 of the normal flow. 4f. In step 4 of the normal flow, if IDE wants to add a related pipeline detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. Use Case resumes on step 1 of the normal flow of Use Case UC-5.1. 2. Use Case returns from UC-5.1 and resumes on step 1 of the normal flow. 4g. In step 4 of the normal flow, if IDE wants to add a related P&ID detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE closes *Instrument List* window. 2. *Instrument List* window closes and the focus returns to the *Main Menu* of the application. 3. Use Case resumes on step 1 of the normal flow of Use Case UC-4.1. 4. Use Case returns from UC-4.1 and resumes on step 1 of the normal flow. 4h. In step 4 of the normal flow, if IDE wants to add an instrument air manifold for an automatic valve but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE selects *Air Manifolds* tab 2. IDE selects *new row*. 3. On the new row, IDE adds the new instrument air manifold with all relevant details. 4. IDE selects *Air Manifold Outputs* tab. 5. IDE selects *new row* and adds new output as many times as

	necessary. Use Case resumes on step 3 of the normal flow.
Exceptions:	N/A
Includes:	N/A
Frequency of Use:	10 000 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	N/A

Use Case ID:	UC-2.3		
Use Case Name:	Modify instrument datasheet		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	5.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: instrumentation design engineer (IDE)		
Description:	IDE modifies detailed instrument data on the instrument datasheet.		
Trigger:	IDE notices that there is some detailed instrument data that must be updated to the instrument datasheet.		
Preconditions:	1. The application is successfully started and the main menu is open. 2. The instrument has been successfully added to the instrument list according to Use Case UC-2.1.		
Post conditions:	1. The updated data will appear on instrument datasheet (and instrument list).		
Normal Flow:	1. IDE selects *Datasheets* from the *Main Menu* of the application. 2. *Instrument Datasheet* window opens. 3. IDE selects the correct instrument datasheet type tab on *Instrument Datasheets* window depending on which instrument type is to be modified. 4. IDE finds the correct instrument to be modified by browsing the datasheets with arrow keys or by typing the instrument tag into the search box. 5. IDE selects either *general data* or *specific data* tab depending on which details IDE wants to modify 6. IDE updates the fields that need to be updated. 7. IDE closes *Instruments Datasheets* window. 8. *Instrument Datasheets* window closes and the focus returns to the *Main Menu* of the application.		
Alternative Flows:	6a. In step 6 of the normal flow, if IDE wants to add document number detail referring to instrument datasheet, hook-up, or location plan, but the document number is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE closes *Instrument Datasheets* window. 2. *Instrument Datasheets* window closes and the focus returns to the *Main Menu* of the application. 3. Use Case resumes on step 1 of the normal flow in Use Case UC-2.2. 4. Use Case returns from UC-2.2 and resumes on step 1 of the normal flow. 6b. In step 6 of the normal flow, if IDE wants to add procurement code		

	detail but the code is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE closes *Instrument Datasheets* window. 2. *Instrument Datasheets* window closes and the focus returns to the *Main Menu* of the application. 3. Use Case resumes on step 1 of the normal flow in Use Case UC-2.2. 4. Use Case returns from UC-2.2 and resumes on step 1 of the normal flow. 6c. In step 6 of the normal flow, if IDE wants to add package unit name detail but the package unit name is not yet shown on the drop-down list from which the correct data should be selected. 1. IDE closes *Instrument Datasheets* window. 2. *Instrument Datasheets* window closes and the focus returns to the *Main Menu* of the application. 3. Use Case resumes on step 1 of the normal flow in Use Case UC-2.2. 4. Use Case returns from UC-2.2 and resumes on step 1 of the normal flow. 6d. In step 6 of the alternative flow, if IDE wants to add package vendor detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE closes *Instrument Datasheets* window. 2. *Instrument Datasheets* window closes and the focus returns to the *Main Menu* of the application. 3. Use Case resumes on step 1 of the normal flow in Use Case UC-2.2. 4. Use Case returns from UC-2.2 and resumes on step 1 of the normal flow. 6e. In step 6 of the normal flow, if IDE wants to add a related process equipment detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE closes *Instrument Datasheets* window. 2. *Instrument Datasheets* window closes and the focus returns to the *Main Menu* of the application. 3. Use Case resumes on step 1 of the normal flow of Use Case UC-4.1. 4. Use Case returns from UC-4.1 and resumes on step 1 of the normal flow. 6f. In step 6 of the normal flow, if IDE wants to add a related pipeline detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE closes *Instrument Datasheets* window. 2. *Instrument Datasheets* window closes and the focus will return to the *Main Menu* of the application. 3. Use Case resumes on step 1 of the normal flow of Use Case UC-5.1. 4. Use Case returns from UC-5.1 and resumes on step 1 of the normal flow. 6g. In step 6 of the normal flow, if IDE wants to add a related P&ID detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE closes *Instrument Datasheets* window. 2. *Instrument Datasheets* window closes and the focus returns to the *Main Menu* of the application. 5. Use Case resumes on step 1 of the normal flow of Use Case UC-4.1.

	6. Use Case returns from UC-4.1 and resumes on step 1 of the normal flow.
Exceptions:	N/A
Includes:	N/A
Frequency of Use:	10 000 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	N/A

Use Case ID:	UC-2.4		
Use Case Name:	Modify instrument process data		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	5.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: process design engineer (PDE)		
Description:	PDE modifies detailed process data on the instrument datasheets.		
Trigger:	PDE notices that there is some detailed process data that must be updated to the instrument datasheet.		
Preconditions:	1. The application is successfully started and the main menu is open. 2. The instrument has been successfully added to the instrument list according to Use Case UC-2.1.		
Post conditions:	1. The updated data will appear on the instrument datasheet.		
Normal Flow:	1. PDE selects *Datasheets* from the *Main Menu* of the application. 2. *Instrument Datasheets* window opens. 3. PDE selects the correct instrument datasheet type tab on *Instrument Datasheets* window depending on which instrument type is to be modified. 4. PDE finds the correct instrument to be modified by browsing the datasheets with arrow keys or by typing the instrument tag into the search box. 5. PDE selects *Process Data* tab. 6. IDE updates the fields that need to be updated. 7. IDE closes *Instrument Datasheets* window. 8. *Instrument Datasheets* window closes and the focus returns to the *Main Menu* of the application.		
Alternative Flows:	N/A		
Exceptions:	N/A		
Includes:	N/A		
Frequency of Use:	10 000 times per project		
Special Requirements:	N/A		
Assumptions:	Users understand English language.		
Notes and Issues:	N/A		

Use Case ID:	UC-3.1		
Use Case Name:	Add new I/O		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	5.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: instrumentation design engineer (IDE)		
Description:	IDE creates a new I/O to the database.		
Trigger:	IDE notices that there is an I/O that must be added to the database.		
Preconditions:	1. The application is successfully started and the main menu is open. 2. The tag of the I/O has been successfully added according to Use Case UC-1.1 where in step 11 of the normal flow, IDE has chosen that *Tag category = I/O*.		
Post conditions:	1. The new I/O appears to the I/O list		
Normal Flow:	1. IDE selects *I/O List* from the *Main Menu* of the application. 2. *I/O List* window opens. 3. IDE clicks *Add New I/O* button. 4. *Add New I/O* window opens. 5. IDE selects the previously added *I/O tag*. 6. IDE clicks *OK.* 7. *Add New I/O* window closes and the focus returns to *I/O List* window. The new I/O appears to *I/O List.*		
Alternative Flows:	6a. In step 6 of the normal flow, if IDE clicks *Cancel* instead of *OK*. 1. The transaction is terminated and changes are not saved. 2. *Add New I/O* window closes and the focus returns to *I/O List* window. 3. Use Case resumes on step 3 of the normal flow.		
Exceptions:	5a. In step 5 of the normal flow, if there are no tags available to choose for due to not added / incorrectly added automation tag. 1. IDE clicks *Cancel* instead of *OK*. 2. *Add New I/O* window closes and the focus returns to *I/O List* window. 3. IDE closes *I/O List* window. 4. Use Case resumes on step 1 of the normal flow of Use Case UC-1.1. 5b. In step 5 of the normal flow, IDE clicks *OK* before the input of all needed information. 1. The application prompts an error message describing the first missing information starting from the top of the form. 2. IDE clicks *OK.* 3. Use Case resumes on step 5 of the normal flow. 5c. In step 1 of the exception 5a, IDE clicks *OK* instead of *Cancel*. 1. Use Case resumes on step 1 of the exception 5b.		
Includes:	UC-1.1 (Add new automation tag)		
Frequency of Use:	1500 times per project		
Special Requirements:	N/A		
Assumptions:	Users understand English language.		
Notes and Issues:	N/A		

3. I/O

Use Case ID:	UC-3.2		
Use Case Name:	Modify I/O list		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	5.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: instrumentation design engineer (IDE)		
Description:	IDE modifies detailed I/O data on the I/O list.		
Trigger:	IDE notices that there is some detailed I/O data that must be updated to the I/O list.		
Preconditions:	1. The application is successfully started and the main menu is open. 2. The I/O has been successfully added to the I/O list according to Use Case UC-3.1.		
Post conditions:	1. The updated data will appear on I/O list (and I/O tab of the instrument list if related to some instrument in the database).		
Normal Flow:	1. IDE selects *I/O List* from the *Main Menu* of the application. 2. *I/O List* window opens. 3. IDE selects *I/O List* tab on *I/O List* window. 4. IDE finds the correct row on the *I/O List* and updates the fields that are needed to be updated. 5. IDE closes *I/O List* window. 6. *I/O List* window closes and the focus returns to the *Main Menu* of the application.		
Alternative Flows:	4a. In step 4 of the normal flow, if IDE wants to add a control system detail but the control system name is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE selects *Control System List* tab from *I/O List* window. 2. IDE adds the new control system name with required details to the end of *Control System List*. 3. Use Case resumes on step 3 of the normal flow. 4b. In step 4 of the normal flow, if IDE wants to add measurement range unit detail but the measurement unit is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE selects *Unit of Measure List list* tab from *I/O List* window. 2. IDE adds the new measurement unit with required details to the end of *Unit of Measure List*. 3. Use Case resumes on step 3 of the normal flow. 4c. In step 4 of the normal flow, if IDE wants to add interlock details. 1. IDE selects *Interlock List with I/Os* tab from *I/O List window*. 2. IDE adds the new interlock with required details, such as interlock tag and related I/O, to the end of *Interlock List with I/Os*. 3. Use Case resumes on step 3 of the normal flow. 4d. In step 2 of the alternative flow, if IDE wants to add interlock tag detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE selects *Interlock List* from *I/O List window*. 2. IDE adds the new interlock tag with required details to the end of *Interlock List*. 3. Use Case resumes on step 1 of the alternative flow 4c. 4e. In step 4 of the normal flow, if IDE wants to add related instrument detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE closes the *I/O List window*.		

	2. *I/O List* window closes and the focus returns to the *Main Menu* of the application. 3. Use Case resumes on step 1 of the normal flow of Use Case UC-2.1. 4. Use Case returns from UC-2.1 and resumes in step 1 of the normal flow. 4f. In step 4 of the normal flow, if IDE wants to add related motor detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE closes the *I/O List window*. 2. *I/O List* window closes and the focus returns to the *Main Menu* of the application. 3. Use Case resumes on step 1 of the normal flow of Use Case UC-6.1. 4. Use Case returns from UC-6.1 and resumes in step 1 of the normal flow.
Exceptions:	N/A
Includes:	N/A
Frequency of Use:	10 000 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	N/A

4. Process Equipment

Use Case ID:	UC-4.1		
Use Case Name:	Modify process equipment details / add new process equipment		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	5.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: process design engineer (PDE)		
Description:	PDE modifies an existing process equipment on the process equipment list or creates a new process equipment to the process equipment list.		
Trigger:	PDE notices that there is a process equipment that must be added to the database or its details must be updated.		
Preconditions:	1. The application is successfully started and the main menu is open.		
Post conditions:	1. The new process equipment appears to the process equipment list or an existing equipment appears as modified.		
Normal Flow:	1. PDE selects *Equipment List* from the *Main Menu* of the application. 2. *Process Equipment List* window opens. 3. PDE selects *Equipment List* tab. 4. PDE selects the row with the process equipment that is to be modified. 5. On the row to be modified, PDE updates all relevant fields. 6. PDE closes the *Process Equipment List* window. 7. *Process Equipment List* closes and the focus returns to the *Main Menu* of the application.		

Alternative Flows:	4a. In step 4 of the normal flow, if PDE notices that the process equipment does not yet exist in the process equipment list. 1. PDE selects *new row*. 2. On the new row, PDE selects the process area, equipment type, and equipment tag number that are compulsory fields. 3. Use Case resumes on step 5 of the normal flow. 4b. In step 2 of the alternative flow 4a, if PDE wants to add a process area detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. PDE closes *Equipment List* window. 2. PDE selects *Automation Tag List* from the *Main Menu* of the application. 3. The *Automation Tag List* window opens. 4. PDE selects *Process Area* tab. 5. PDE adds the new process area to the end of the list. 6. PDE closes the *Automation Tag List* window. 7. *Automation Tag List* window closes and the focus returns to the *Main Menu* of the application. 8. Use Case resumes on step 1 of the normal flow. 4c. In step 2 of the alternative flow 4a, if PDE wants to add an equipment type detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. PDE opens *Equipment Type* tab from the *Process Equipment List* window. 2. PDE clicks *new row*. 3. The focus goes on the new row. 4. On the new row, PDE adds the new equipment type with all relevant information. 5. Use Case resumes on step 3 of the normal flow. 5a. In step 5 of the normal flow, if PDE wants to add a P&ID detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. PDE opens *P&ID* tab from the *Process Equipment List* window. 2. PDE clicks *new row*. 3. The focus goes on the new row. 4. On the new row, PDE adds the new P&ID with all relevant information. 5. Use Case resumes on step 3 of the normal flow.
Exceptions:	6a. In step 6 of the normal flow, if PDE tries to close the *Process Equipment List* window even though not all compulsory fields are filled in. 1. A new window opens with a warning message indicating that all compulsory fields are not filled in. 2. PDE clicks *OK*. 3. Use Case resumes on step 2 of the alternative flow 4a.
Includes:	N/A
Frequency of Use:	1000 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	N/A

Use Case ID:	UC-4.2		
Use Case Name:	Delete process equipment		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A

Date Created:	6.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: process design engineer (PDE)		
Description:	PDE marks a process equipment as *deleted* on the database.		
Trigger:	PDE notices that there is a process equipment that must be deleted from the process equipment list		
Preconditions:	1. The application is successfully started and the main menu is open.		
Post conditions:	1. The process equipment disappears from the process equipment list		
Normal Flow:	1. PDE selects *Equipment List* from the *Main Menu* of the application. 2. *Process Equipment List* window opens. 3. PDE selects *Equipment List* tab. 4. PDE selects the row that he wants to remove from the list. 5. On the same row, PDE checks the *deleted* check box. 6. PDE closes the *Process Equipment List* window. 7. *Process Equipment List* window closes and the focus returns to the *Main Menu* of the application.		
Alternative Flows:	N/A		
Exceptions:	N/A		
Includes:	N/A		
Frequency of Use:	100 times per project		
Special Requirements:	N/A		
Assumptions:	Users understand English language.		
Notes and Issues:	If PDE wants to cancel the deletion, PDE must return to step 1 of the normal flow and uncheck the *deleted* check box in step 3 of the normal flow. If deleted fields are not shown, PDE must check the check box *Show deleted process equipment* on *Settings* window.		

5. Pipeline

Use Case ID:	UC-5.1		
Use Case Name:	Modify pipeline details / add new pipeline		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	6.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: process design engineer (PDE)		
Description:	PDE modifies an existing pipeline on the pipeline list or creates a new pipeline to the pipeline list.		
Trigger:	PDE notices that there is a pipeline that must be added to the database or its details must be updated.		
Preconditions:	1. The application is successfully started and the main menu is open.		
Post conditions:	1. The new pipeline appears to the pipeline list or an existing pipeline appears as modified.		
Normal Flow:	1. PDE selects *Line List* from the *Main Menu* of the application. 2. *Pipeline List* window opens. 3. PDE selects *Pipeline List* tab. 4. PDE selects the row with the pipeline that is to be modified. 5. On the row to be modified, PDE updates all relevant fields. 6. PDE closes the *Pipeline List* window. 7. *Pipeline List* closes and the focus returns to the *Main Menu* of the application.		

Alternative Flows:	4a. In step 4 of the normal flow, if PDE notices that the pipeline does not yet exist in the process equipment list. 1. PDE selects *new row*. 2. On the new row, PDE selects the process area, process fluid, and pipeline tag number that are compulsory fields. 3. Use Case resumes on step 5 of the normal flow. 4b. In step 2 of the alternative flow, if PDE wants to add a process area detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. PDE closes *Pipeline List* window. 2. PDE selects *Automation Tag List* from the *Main Menu* of the application. 3. The *Automation Tag List* window opens. 4. PDE selects *Process Area* tab. 5. PDE adds the new process area to the end of the list. 6. PDE closes the *Automation Tag List* window. 7. *Automation Tag List* window closes and the focus returns to the *Main Menu* of the application. 8. Use Case resumes on step 1 of the normal flow. 4c. In step 2 of the alternative flow 4a, if PDE wants to add a process fluid detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. PDE opens *Process Fluid* tab from the *Pipeline List* window. 2. PDE clicks *new row*. 3. The focus goes on the new row. 4. On the new row, PDE adds the new process fluid with all relevant information. 5. Use Case resumes on step 3 of the normal flow. 4d. In step 2 of the alternative flow 4a, if PDE wants to add a pipeline number detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. PDE opens *Pipeline Number* tab from the *Pipeline List* window. 2. PDE clicks *new row*. 3. The focus goes on the new row. 4. On the new row, PDE adds the new pipeline number with all relevant information. 5. Use Case resumes on step 3 of the normal flow. 5a. In step 5 of the normal flow, if PDE wants to add a P&ID detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. PDE closes *Pipeline List* window. 2. *Pipeline List* window closes and the focus returns to the *Main Menu* of the application. 3. PDE selects *Equipment List* from the *Main Menu* of the application. 4. *Process Equipment List* window opens. 5. PDE opens *P&ID* tab from the *Process Equipment List* window. 6. PDE clicks *new row*. 7. The focus goes on the new row. 8. On the new row, PDE adds the new P&ID with all relevant information. 9. PDE closes *Process Equipment List* window. 10. *Process Equipment List* closes and the focus returns to the *Main Menu* of the application. 11. Use Case resumes on step 1 of the normal flow. 5b. In step 5 of the normal flow, if PDE wants to add an insulation detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. PDE opens *Insulation* tab from the *Pipeline List* window. 2. PDE clicks *new row*.

	3. The focus goes on the new row. 4. On the new row, PDE adds the new insulation type with all relevant information. 5. Use Case resumes on step 3 of the normal flow. 5c. In step 5 of the normal flow, if PDE wants to add a nominal diameter detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. PDE opens *Diameter list* tab from the *Pipeline List* window. 2. PDE clicks *new row*. 3. The focus goes on the new row. 4. On the new row, PDE adds the new nominal diameter with all relevant information. 5. Use Case resumes on step 3 of the normal flow. 5c. In step 5 of the normal flow, if PDE wants to add a heat tracing detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. PDE opens *Heat Tracing* tab from the *Pipeline List* window. 2. PDE clicks *new row*. 3. The focus goes on the new row. 4. On the new row, PDE adds the new heat tracing type with all relevant information. 5. Use Case resumes on step 3 of the normal flow. 5d. In step 5 of the normal flow, if PDE wants to add a pipeline specification detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. PDE opens *Pipeline Specification* tab from the *Pipeline List* window. 2. PDE clicks *new row*. 3. The focus goes on the new row. 4. On the new row, PDE adds the new pipeline specification type with all relevant information. Use Case resumes on step 3 of the normal flow.
Exceptions:	6a. In step 6 of the normal flow, if PDE tries to close the *Pipeline List* window even though not all compulsory fields are filled in. 1. A new window opens with a warning message indicating that all compulsory fields are not filled in. 2. PDE clicks *OK*. 3. Use Case resumes on step 2 of the alternative flow 4a.
Includes:	N/A
Frequency of Use:	1000 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	N/A

Use Case ID:	UC-5.2		
Use Case Name:	Delete pipeline		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	6.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: process design engineer (PDE)		
Description:	PDE marks a pipeline as *deleted* on the database.		
Trigger:	PDE notices that there is a pipeline that must be deleted from the process equipment list		

Preconditions:	1. The application is successfully started and the main menu is open.
Post conditions:	1. The process equipment disappears from the pipeline list
Normal Flow:	1. PDE selects L*ine List* from the *Main Menu* of the application. 2. *Pipeline List* window opens. 3. PDE selects *Pipeline List* tab. 4. PDE selects the row that he wants to remove from the list. 5. On the same row, PDE checks the *deleted* check box. 6. PDE closes the *Pipeline List* window. 7. *Pipeline List* closes and the focus returns to the *Main Menu* of the application.
Alternative Flows:	N/A
Exceptions:	N/A
Includes:	N/A
Frequency of Use:	100 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	If PDE wants to cancel the deletion, PDE must return to step 1 of the normal flow and uncheck the *deleted* check box in step 3 of the normal flow. If deleted fields are not shown, PDE must check the check box *Show deleted pipelines* on *Settings* window.

6. Motor

Use Case ID:	UC-6.1		
Use Case Name:	Modify motor details / add new motor		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	6.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: electrical design engineer (EDE)		
Description:	EDE modifies an existing motor on the motor list or creates a new motor to the motor list.		
Trigger:	EDE notices that there is a motor that must be added to the database or its details must be updated.		
Preconditions:	1. The application is successfully started and the main menu is open.		
Post conditions:	1. The new motor appears to the motor list or an existing motor appears as modified.		
Normal Flow:	1. EDE selects *Motor List* from the *Main Menu* of the application. 2. *Motor List* window opens. 3. EDE selects *Motor List* tab. 4. EDE selects the row with the motor that is to be modified. 5. On the row to be modified, EDE updates all relevant fields. 6. EDE closes the *Motor List* window. 7. *Motor List* window closes and the focus returns to the *Main Menu* of the application.		
Alternative Flows:	4a. In step 4 of the normal flow, if EDE notices that the motor does not yet exist in the motor list. 1. EDE selects *new row*. 2. On the new row, EDE adds the motor tag, which is a compulsory field, and all other non-compulsory fields. 3. Use Case resumes on step 5 of the normal flow. 5a. In step 5 of the normal flow, if EDE wants to add a feeder panel		

	detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. EDE opens *Feeder Panel List* tab from the *Motor List* window. 2. EDE clicks *new row*. 3. The focus goes on the new row. 4. On the new row, EDE adds the new feeder panel with all relevant information. 5. Use Case resumes on step 3 of the normal flow. 5b. In step 5 of the normal flow, if EDE wants to add a related process equipment detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. EDE closes the *Motor List* window. 2. *Motor List* closes and the focus returns to the *Main Menu* of the application. 3. Use Case resumes on step 1 of the normal flow of Use Case UC-4.1. 4. Use Case returns from Use Case UC-4.1 and resumes on step 1 of normal flow.
Exceptions:	6a. In step 6 of the normal flow, if EDE tries to close the *Motor List* window even though not all compulsory fields are filled in. 1. A new window opens with a warning message indicating that all compulsory fields are not filled in. 2. EDE clicks *OK*. 3. Use Case resumes on step 2 of the alternative flow 4a.
Includes:	N/A
Frequency of Use:	300 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	N/A

Use Case ID:	UC-6.2		
Use Case Name:	Delete motor		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	6.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: electrical design engineer (EDE)		
Description:	EDE marks a motor as *deleted* on the database.		
Trigger:	EDE notices that there is a motor that must be deleted from the motor list		
Preconditions:	1. The application is successfully started and the main menu is open.		
Post conditions:	1. The motor disappears from the motor list		
Normal Flow:	1. EDE selects *Motor List* from the *Main Menu* of the application. 2. *Motor List* window opens. 3. EDE selects *Motor List* tab. 4. EDE selects the row that he wants to remove from the list. 5. On the same row, EDE checks the *deleted* check box. 6. EDE closes the *Motor list* window. 7. *Motor list* closes and the focus returns to the *Main Menu* of the application.		
Alternative Flows:	N/A		

Exceptions:	N/A
Includes:	N/A
Frequency of Use:	10 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	If EDE wants to cancel the deletion, EDE must return to step 1 of the normal flow and uncheck the *deleted* check box in step 3 of the normal flow. If deleted fields are not shown, EDE must check the check box *Show deleted motors* on the *Settings* window.

7. Cable

Use Case ID:	UC-7.1		
Use Case Name:	Modify cable details / add new cable		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	7.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: Instrumentation design engineer (IDE)		
Description:	IDE modifies an existing cable on the cable list or creates a new cable to the cable list.		
Trigger:	IDE notices that there is a cable that must be added to the database or its details must be updated.		
Preconditions:	1. The application is successfully started and the main menu is open.		
Post conditions:	1. The new cable appears to the cable list or an existing cable appears as modified.		
Normal Flow:	1. IDE selects *Cable List* from the *Main Menu* of the application. 2. *Cable List* window opens. 3. IDE selects *Cable List* tab. 4. IDE selects the row with the cable that is to be modified. 5. On the row to be modified, IDE updates all relevant fields. 6. IDE closes the *Cable List* window. 7. *Cable list* closes and the focus returns to the *Main Menu* of the application.		
Alternative Flows:	4a. In step 4 of the normal flow, if IDE notices that the cable does not yet exist in the cable list. 1. IDE selects *new row*. 2. On the new row, IDE adds the cable category, which is a compulsory field, and all other non-compulsory fields. 3. Use Case resumes on step 5 of the normal flow. 5a. In step 5 of the normal flow, if IDE wants to add a cable type detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE opens *Cable Type List* tab from the *Cable List* window. 2. IDE clicks *new row*. 3. The focus goes on the new row. 4. On the new row, IDE adds the new cable type with all relevant information. 5. Use Case resumes on step 3 of the normal flow. 5b. In step 5 of the normal flow, if IDE wants to add an electrical enclosure terminal row detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE opens *Electrical Enclosure Terminal Row List* in the *Cable List* window. 2. IDE clicks *new row*. 3. The focus goes on the new row.		

	4. On the new row, IDE adds the electrical enclosure terminal row with all relevant information. 5. Use Case resumes on step 3 of the normal flow. 5c. In step 4 of the alternative flow 5b, if IDE wants to add an electrical enclosure detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE opens *Electrical Enclosure List* in the *Cable List* window. 2. IDE clicks *new row*. 3. The focus goes on the new row. 4. On the new row, IDE adds the electrical enclosure with all relevant information. 5. Use Case resumes on step 3 of the normal flow. 5d. In step 5 of the normal flow, if IDE wants to add a related instrument detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE closes the *Cable List* window. 2. *Cable List* closes and the focus returns to the *Main Menu* of the application. 3. Use Case resumes on step 1 of the normal flow of Use Case UC-2.1. 4. Use Case returns from Use Case UC-2.1 and resumes on step 1 of normal flow. 5e. In step 5 of the normal flow, if IDE wants to add a related motor detail but it is not yet shown in the drop-down list from which the correct data should be selected. 1. IDE closes the *Cable List* window. 2. *Cable List* closes and the focus returns to the *Main Menu* of the application. 3. Use Case resumes on step 1 of the normal flow of Use Case UC-6.1. 4. Use Case returns from Use Case UC-6.1 and resumes on step 1 of normal flow.
Exceptions:	6a. In step 6 of the normal flow, if IDE tries to close the *Cable List* window even though not all compulsory fields are filled in. 1. A new window opens with a warning message indicating that all compulsory fields are not filled in. 2. IDE clicks *OK*. 3. Use Case resumes on step 2 of the alternative flow 4a.
Includes:	N/A
Frequency of Use:	2000 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	N/A

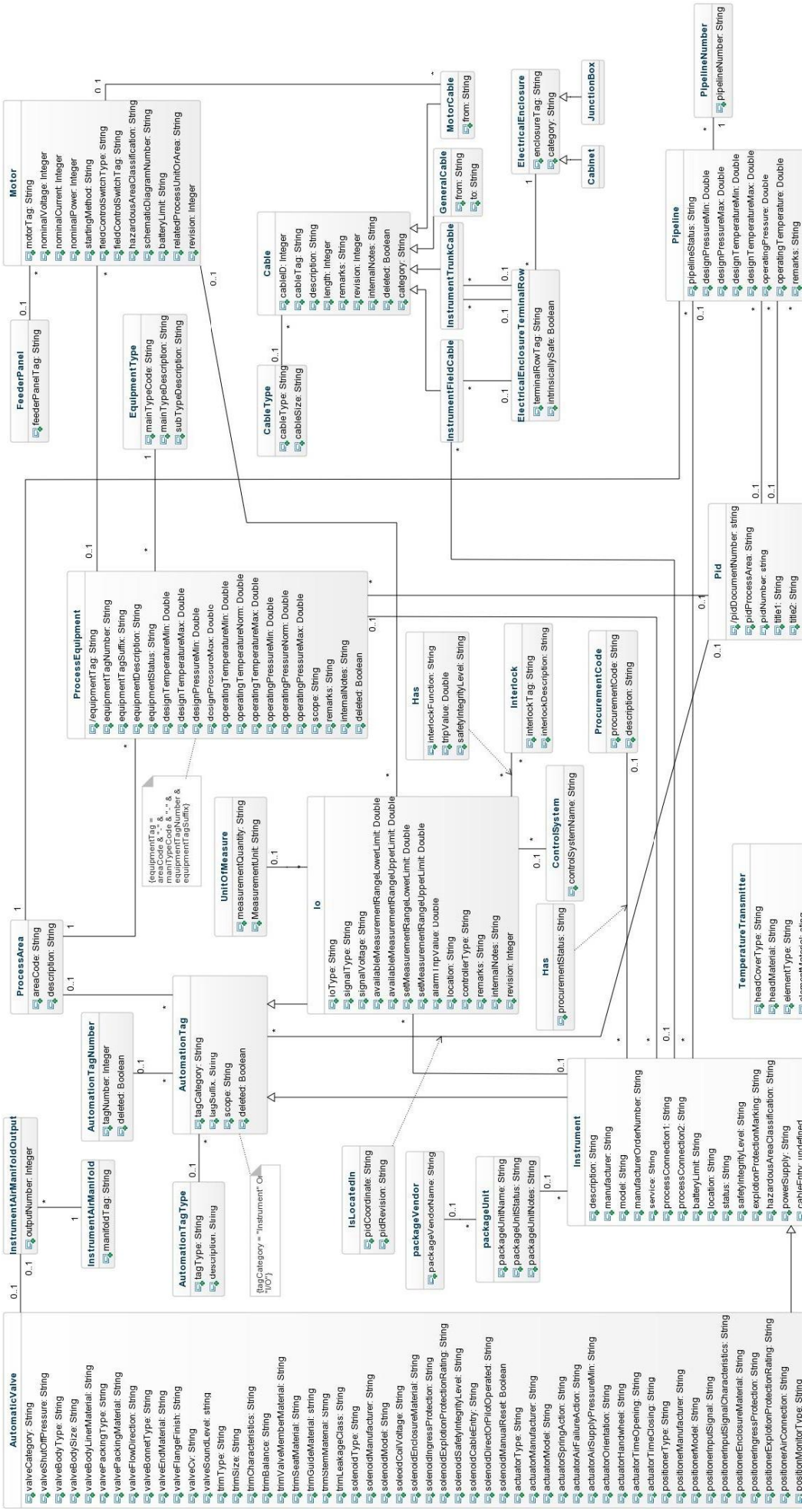
Use Case ID:	UC-7.2		
Use Case Name:	Delete cable		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	7.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actor: instrumentation design engineer (IDE)		
Description:	IDE marks a cable as *deleted* on the database.		
Trigger:	IDE notices that there is a cable that must be deleted from the cable list		
Preconditions:	1. The application is successfully started and the main menu is open.		

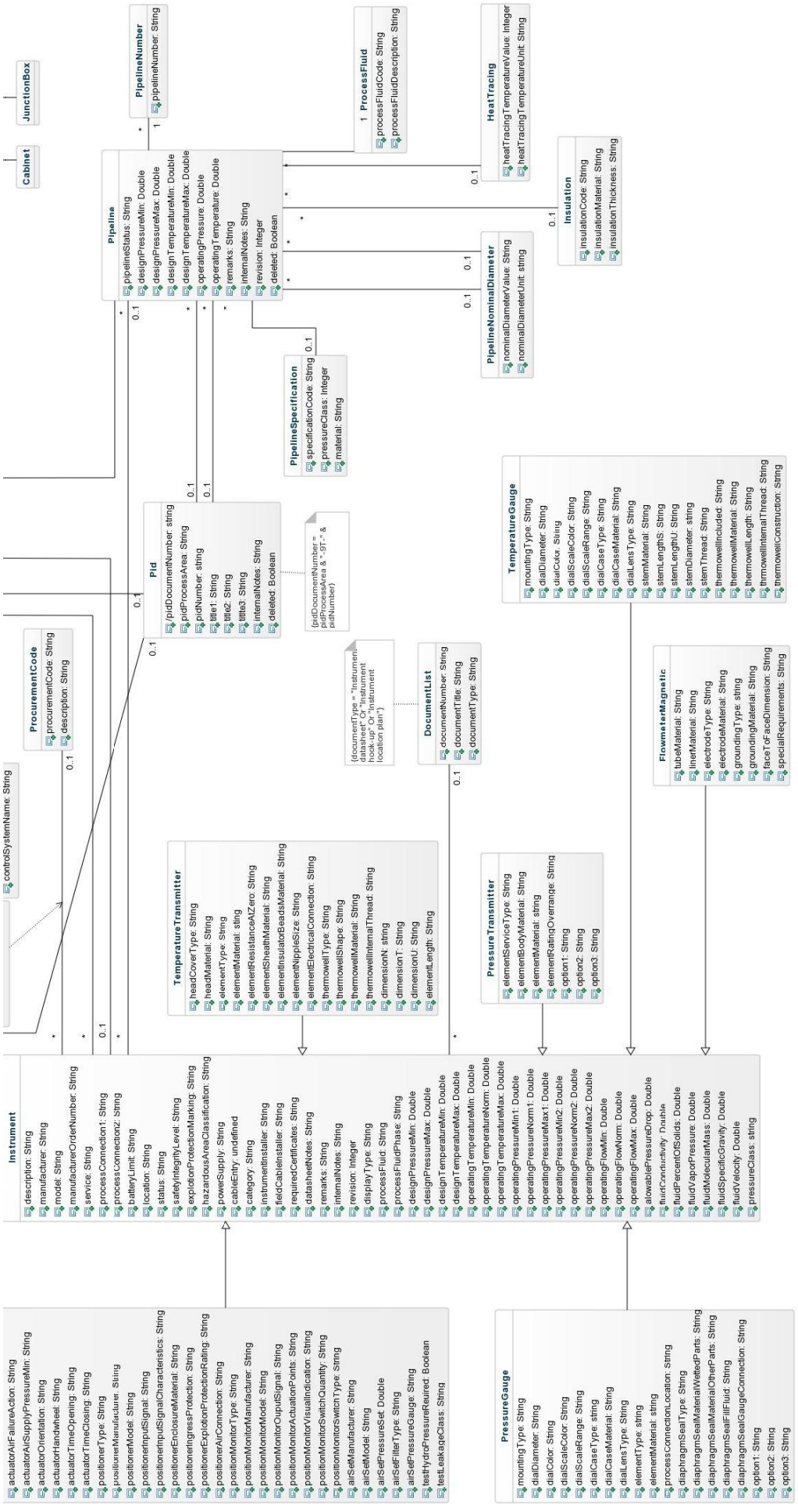
Post conditions:	1. The cable disappears from the cable list.
Normal Flow:	1. IDE selects *Cable List* from the *Main Menu* of the application. 2. *Cable List* window opens. 3. IDE selects *Cable List* tab. 4. IDE selects the row that he wants to remove from the list. 5. On the same row, IDE checks the *deleted* check box. 6. IDE closes the *Cable List* window. 7. *Cable List* closes and the focus returns to the *Main Menu* of the application.
Alternative Flows:	N/A
Exceptions:	N/A
Includes:	N/A
Frequency of Use:	100 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	If IDE wants to cancel the deletion, IDE must return to step 1 of the normal flow and uncheck the *deleted* check box in step 3 of the normal flow. If deleted fields are not shown, IDE must check the check box *Show deleted cables* on the *Settings* window.

8. Reports

Use Case ID:	UC-8.1		
Use Case Name:	Print report for: pipeline list / equipment list / I/O list / instrument list / instrument datasheets / cable list / motor list		
Created By:	Mika Latva-Kyyny	**Last Updated By:**	N/A
Date Created:	7.12.2015	**Last Revision Date:**	N/A
Actors:	Primary actors: instrumentation design engineer (IDE), process design engineer (PDE), electrical design engineer (EDE)		
Description:	IDE, PDE, or EDE prints a report from the database.		
Trigger:	IDE, PDE, or EDE needs to print a printed report for viewing or for delivering it to the client.		
Preconditions:	1. The application is successfully started and the main menu is open.		
Post conditions:	1. The report is exported from the database in PDF format.		
Normal Flow:	1. User (IDE, PDE, or EDE) selects *Reports* from the *Main Menu* of the application. 2. *Publish report* window opens. 3. User selects the type of report that he wants to print by clicking. *4.* *Publish report* window closes and a new window opens showing the report in print preview mode. 5. User right-mouse clicks the report and selects *Print.* 6. *Print* window opens. 7. User chooses the printer settings he wants to use and click *OK*. 8. The *Print* window closes and the report is printed. 9. User closes the report window. 10. Report window closes and the focus returns to the *Main Menu* of the application.		
Alternative Flows:	7a. In step 7 of the normal flow, if user clicks *Cancel* instead of *OK*. 1. *Print* window closes. 2. Use Case resumes on step 5 of the normal flow.		

Exceptions:	N/A
Includes:	N/A
Frequency of Use:	200 times per project
Special Requirements:	N/A
Assumptions:	Users understand English language.
Notes and Issues:	N/A

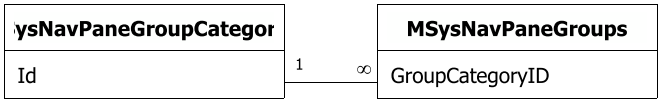

AutomaticValve
valveCategory: String
valveShutOffPressure: String
valveBodyType: String
valveBodySize: String
valveBodyLinerMaterial: String
valvePackingType: String
valvePackingMaterial: String
valveFlowDirection: String
valveBonnetType: String
valveEndMaterial: String
valveFlangeFinish: String
valveCv: String
valveSoundLevel: string
trimType: String
trimSize: String
trimCharacteristics: String
trimBalance: String
trimValveMemberMaterial: String
trimSeatMaterial: String
trimGuideMaterial: string
trimStemMaterial: string
trimLeakageClass: String
solenoidType: String
solenoidManufacturer: String
solenoidModel: String
soleoidCoilVoltage: String
solenoidEnclosureMaterial: String
solenoidIngressProtection: String
solenoidExplotionProtectionRating: String
solenoidSafetyIntegrityLevel: String
solenoidCableEntry: String
solenoidDirectOrPilotOperated: String
solenoidManualReset: Boolean
actuatorType: String
actuatorManufacturer: String
actuatorModel: String
actuatorSpringAction: String
actuatorAirFailureAction: String
actuatorAirSupplyPressureMin: String
actuatorOrientation: String
actuatorHandwheel: String
actuatorTimeOpening: String
actuatorTimeClosing: String
positionerType: String
positionerManufacturer: String
positionerModel: String
positionerInputSignal: String
positionerInputSignalCharacteristics: String
positionerEnclosureMaterial: String
positionerIngressProtection: String
positionerExplotionProtectionRating: String
positionerAirConnection: String
positionerMonitorType: String
InstrumentAirManifoldOutput
outputNumber: Integer
InstrumentAirManifold
manifoldTag: String
AutomationTagNumber
tagNumber: Integer
deleted: Boolean
AutomationTagType
tagType: String
description: String
AutomationTag
tagCategory: String
tagSuffix: String
scope: String
deleted: Boolean
{tagCategory = "Instrument" Or "I/O"}
ProcessArea
areaCode: String
description: String
{equipmentTag = areaCode &"-"& mainTypeCode &"-"& equipmentTagNumber & equipmentTagSuffix}
ProcessEquipment
/equipmentTag: String
equipmentTagNumber: String
equipmentTagSuffix: String
equipmentDescription: String
equipmentStatus: String
designTemperatureMin: Double
designTemperatureMax: Double
designPressureMin: Double
designPressureMax: Double
operatingTemperatureMin: Double
operatingTemperatureNorm: Double
operatingTemperatureMax: Double
operatingPressureMin: Double
operatingPressureNorm: Double
operatingPressureMax: Double
scope: String
remarks: String
internalNotes: String
deleted: Boolean
UnitOfMeasure
measurementQuantity: String
MeasurementUnit: String
Io
ioType: String
signalType: String
signalVoltage: String
availableMeasurementRangeLowerLimit: Double
availableMeasurementRangeUpperLimit: Double
setMeasurementRangeLowerLimit: Double
setMeasurementRangeUpperLimit: Double
alarmInpValue: Double
location: String
controllerType: String
remarks: String
internalNotes: String
revision: Integer
Has
interlockFunction: String
tripValue: Double
safetyIntegrityLevel: String
Interlock
interlockTag: String
interlockDescription: String
ControlSystem
controlSystemName: String
Has
procurementStatus: String
ProcurementCode
procurementCode: String
description: String
IsLocatedIn
pidCoordinate: String
pidRevision: String
packageVendor
packageVendorName: String
packageUnit
packageUnitName: String
packageUnitStatus: String
packageUnitNotes: String
Instrument
description: String
manufacturer: String
model: String
manufacturerOrderNumber: String
service: String
processConnection1: String
processConnection2: String
batteryLimit: String
location: String
status: String
safetyIntegrityLevel: String
explotionProtectionMarking: String
hazardousAreaClassification: String
powerSupply: String
cableEntry: undefined
TemperatureTransmitter
headCoverType: String
headMaterial: String
elementType: String
Pid
/pidDocumentNumber: string
pidProcessArea: String
pidNumber: string
title1: String
title2: String
Pipeline
pipelineStatus: String
designPressureMin: Double
designPressureMax: Double
designTemperatureMin: Double
designTemperatureMax: Double
operatingPressure: Double
operatingTemperature: Double
remarks: String
PipelineNumber
pipelineNumber: String
Motor
motorTag: String
nominalVoltage: Integer
nominalCurrent: Integer
nominalPower: Integer
startingMethod: String
fieldControlSwitchType: String
fieldControlSwitchTag: String
hazardousAreaClassification: String
schematicDiagramNumber: String
batteryLimit: String
relatedProcessUnitOrArea: String
revision: Integer
FeederPanel
feederPanelTag: String
EquipmentType
mainTypeCode: String
mainTypeDescription: String
subTypeDescription: String
Cable
cableID: Integer
cableTag: String
description: String
length: Integer
remarks: String
revision: Integer
internalNotes: String
deleted: Boolean
category: String
CableType
cableType: String
cableSize: String
MotorCable
from: String
GeneralCable
from: String
to: String
InstrumentTrunkCable
InstrumentFieldCable
ElectricalEnclosureTerminalRow
terminalRowTag: String
intrinsicallySafe: Boolean
ElectricalEnclosure
enclosureTag: String
category: String
JunctionBox
Cabinet


JunctionBox
Cabinet
PipelineNumber
pipelineNumber: String
Pipeline
pipelineStatus: String
designPressureMin: Double
designPressureMax: Double
designTemperatureMin: Double
designTemperatureMax: Double
operatingPressure: Double
operatingTemperature: Double
remarks: String
internalNotes: String
revision: Integer
deleted: Boolean
1 ProcessFluid
processFluidCode: String
processFluidDescription: String
HeatTracing
heatTracingTemperatureValue: Integer
heatTracingTemperatureUnit: String
Insulation
insulationCode: String
insulationMaterial: String
insulationThickness: String
PipelineNominalDiameter
nominalDiameterValue: String
nominalDiameterUnit: string
PipelineSpecification
specificationCode: String
pressureClass: Integer
material: String
Pid
{pidDocumentNumber: string
pidProcessArea: String
pidNumber: string
title1: String
title2: String
title3: String
internalNotes: String
deleted: Boolean
{pidDocumentNumber = pidProcessArea &"-SFI-" & pidNumber}
{documentType = "Instrument datasheet" Or "Instrument hook-up" Or "Instrument location plan"}
DocumentList
documentNumber: String
documentTitle: String
documentType: String
ProcurementCode
procurementCode: String
description: String
controlSystemName: String
TemperatureTransmitter
headCoverType: String
headMaterial: String
elementType: String
elementMaterial: sting
elementResistanceAtZero: String
elementSheathMaterial: String
elementInsulatorBeadsMaterial: String
elementNippleSize: String
elementElectricalConnection: String
thermowellType: String
thermowellShape: String
thermowellMaterial: String
thermowellInternalThread: String
dimensionN: string
dimensionT: String
dimensionU: String
elementLength: String
PressureTransmitter
elementServiceType: String
elementBodyMaterial: String
elementMaterial: string
elementRatingOverrange: String
option1: String
option2: String
option3: String
TemperatureGauge
mountingType: String
dialDiameter: String
dialColor: String
dialScaleColor: String
dialScaleRange: String
dialCaseType: String
dialCaseMaterial: String
dialLensType: String
stemMaterial: String
stemLengthS: String
stemLengthU: String
stemDiameter: string
stemThread: String
thermowellincluded: String
thermowellMaterial: String
thermowellLength: String
thermowellInternalThread: String
thermowellConstruction: String
FlowmeterMagnetic
tubeMaterial: String
linerMaterial: String
electrodeType: String
electrodeMaterial: String
groundingType: string
groundingMaterial: String
faceToFaceDimension: String
specialRequirements: String
Instrument
description: String
manufacturer: String
model: String
manufacturerOrderNumber: String
service: String
processConnection1: String
processConnection2: String
batteryLimit: String
location: String
status: String
safetyIntegrityLevel: String
explotionProtectionMarking: String
hazardousAreaClassification: String
powerSupply: String
cableEntry: undefined
category: String
instrumentInstaller: String
fieldCableInstaller: String
requiredCertificates: String
datasheetNotes: String
remarks: String
internalNotes: String
revision: Integer
displayType: String
processFluid: String
processFluidPhase: String
designPressureMin: Double
designPressureMax: Double
designTemperatureMax: Double
designTemperatureMin: Double
operatingTemperatureNorm: Double
operatingTemperatureMax: Double
operatingPressureMin1: Double
operatingPressureNorm1: Double
operatingPressureMax1: Double
operatingPressureMin2: Double
operatingPressureNorm2: Double
operatingPressureMax2: Double
operatingFlowMin: Double
operatingFlowNorm: Double
operatingFlowMax: Double
allowablePressureDrop: Double
fluidConductivity: Double
fluidPercentOfSolids: Double
fluidVaporPressure: Double
fluidMolecularMass: Double
fluidSpecificGravity: Double
fluidVelocity: Double
pressureClass: string
actuatorAirFailureAction: String
actuatorAirSupplyPressureMin: String
actuatorOrientation: String
actuatorHandwheel: String
actuatorTimeOpening: String
actuatorTimeClosing: String
positionerType: String
positionerManufacturer: String
positionerModel: String
positionerInputSignal: String
positionerInputSignalCharacteristics: String
positionerEnclosureMaterial: String
positionerIngressProtection: String
positionerExplotionProtectionRating: String
positionerAirConnection: String
positionerMonitorType: String
positionMonitorManufacturer: String
positionMonitorModel: String
positionMonitorOuputSignal: String
positionMonitorActuationPoints: String
positionMonitorVisualIndication: String
positionMonitorSwitchQuantity: String
positionMonitorswitchType: String
airSetManufacturer: String
airSetModel: String
airSetPressureSet: Double
airSetFilterType: String
airSetPressureGauge: String
testHydroPressureReuired: Boolean
testLeakageClass: String
PressureGauge
mountingType: String
dialDiameter: String
dialColor: String
dialScaleColor: String
dialScaleRange: String
dialCaseType: string
dialCaseMaterial: String
dialLensType: String
elementType: string
elementMaterial: string
processConnectionLocation: String
diaphragmSealType: String
diaphragmSealMaterialWettedParts: String
diaphragmSealMaterialOtherParts: String
diaphragmSealFillFluid: String
diaphragmSealGaugeConnection: String
option1: String
option2: String
option3: String

APPENDIX E

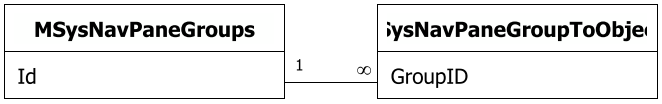
Relationships

MSysNavPaneGroupCategoriesMSysNavPaneGroups



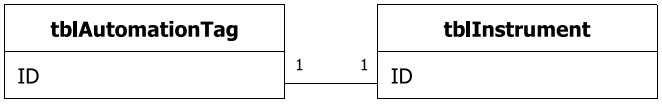
Attributes: Enforced; Cascade Updates; Cascade Deletes
RelationshipType: One-To-Many

MSysNavPaneGroupsMSysNavPaneGroupToObjects



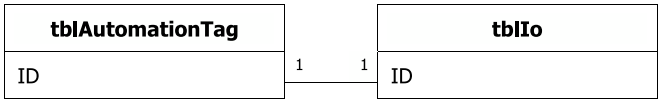
Attributes: Enforced; Cascade Updates; Cascade Deletes
RelationshipType: One-To-Many

tblAutomationTagtblInstrument



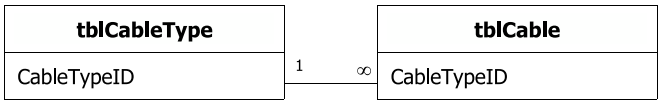
Attributes: Unique; Enforced
RelationshipType: One-To-One

tblAutomationTagtblIo



Attributes: Unique; Enforced
RelationshipType: One-To-One

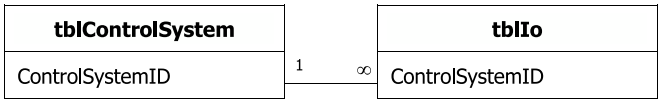
tblCableTypetblCable



Attributes: Enforced
RelationshipType: One-To-Many

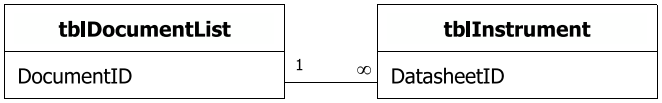
APPENDIX E

tblControlSystemtblIo



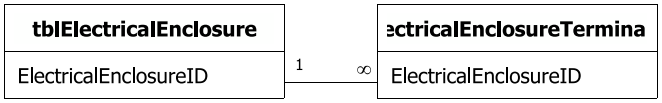
Attributes: Enforced
RelationshipType: One-To-Many

tblDocumentListtblInstrument



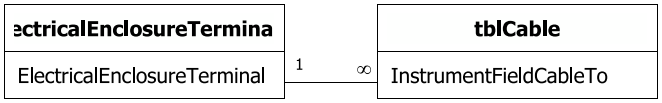
Attributes: Enforced
RelationshipType: One-To-Many

tblElectricalEnclosuretblElectricalEnclosureTerminalRow



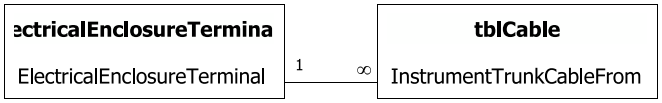
Attributes: Enforced
RelationshipType: One-To-Many

tblElectricalEnclosureTerminalRowtblCable



Attributes: Enforced
RelationshipType: One-To-Many

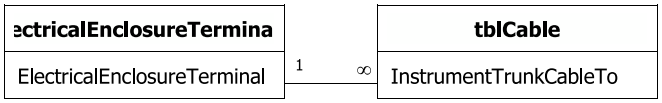
tblElectricalEnclosureTerminalRowtblCable



Attributes: Enforced
RelationshipType: One-To-Many

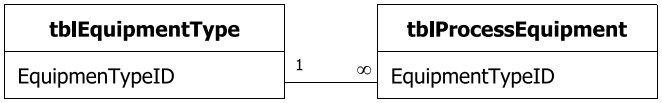
APPENDIX E

tblElectricalEnclosureTerminalRowtblCable



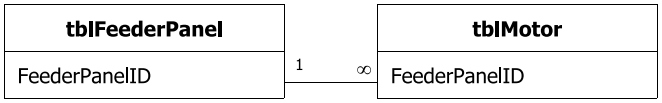
Attributes: Enforced
RelationshipType: One-To-Many

tblEquipmentTypetblProcessEquipment



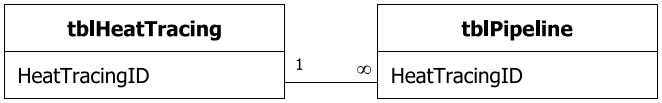
Attributes: Enforced
RelationshipType: One-To-Many

tblFeederPaneltblMotor



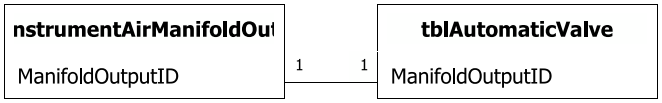
Attributes: Enforced
RelationshipType: One-To-Many

tblHeatTracingtblPipeline



Attributes: Enforced
RelationshipType: One-To-Many

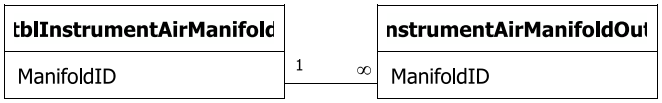
tblInstrumentAirManifoldOutputtblAutomaticValve



Attributes: Unique; Enforced
RelationshipType: One-To-One

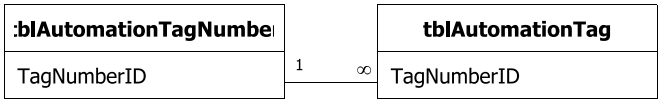
APPENDIX E

tblInstrumentAirManifoldtblInstrumentAirManifoldOutput



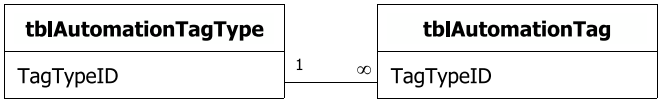
Attributes: Enforced
RelationshipType: One-To-Many

tblAutomationTagNumbertblAutomationTag



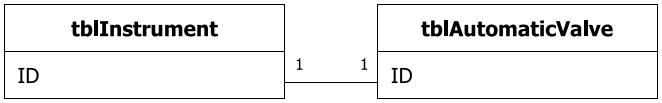
Attributes: Enforced
RelationshipType: One-To-Many

tblAutomationTagTypetblAutomationTag



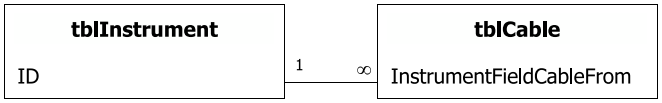
Attributes: Enforced
RelationshipType: One-To-Many

tblInstrumenttblAutomaticValve



Attributes: Unique; Enforced
RelationshipType: One-To-One

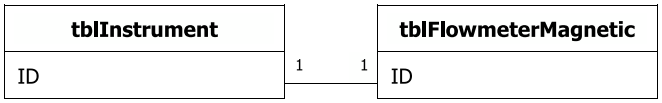
tblInstrumenttblCable



Attributes: Enforced
RelationshipType: One-To-Many

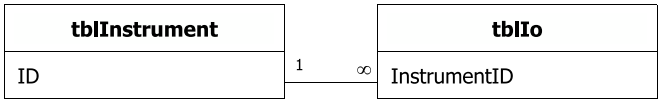
APPENDIX E

tblInstrumenttblFlowmeterMagnetic



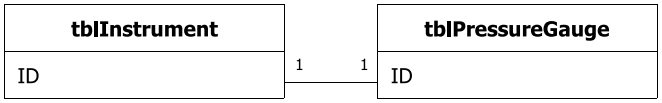
Attributes: Unique; Enforced
RelationshipType: One-To-One

tblInstrumenttblIo



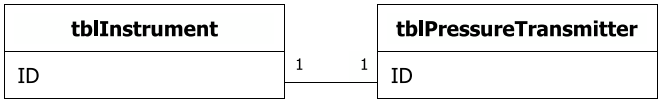
Attributes: Enforced
RelationshipType: One-To-Many

tblInstrumenttblPressureGauge



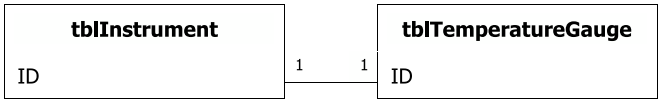
Attributes: Unique; Enforced
RelationshipType: One-To-One

tblInstrumenttblPressureTransmitter



Attributes: Unique; Enforced
RelationshipType: One-To-One

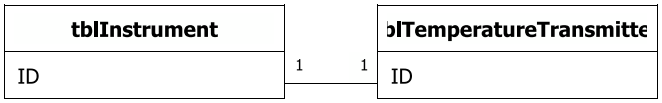
tblInstrumenttblTemperatureGauge



Attributes: Unique; Enforced
RelationshipType: One-To-One

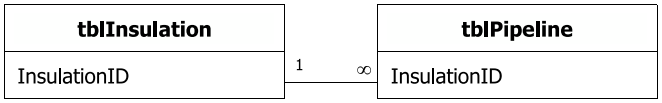
APPENDIX E

tblInstrumenttblTemperatureTransmitter



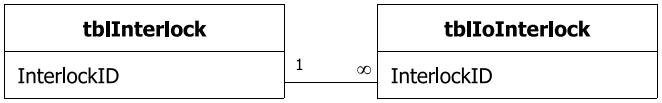
Attributes: Unique; Enforced
RelationshipType: One-To-One

tblInsulationtblPipeline



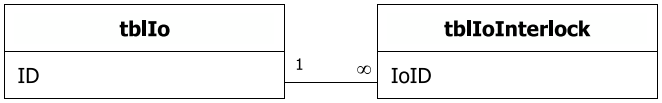
Attributes: Enforced
RelationshipType: One-To-Many

tblInterlocktblIoInterlock



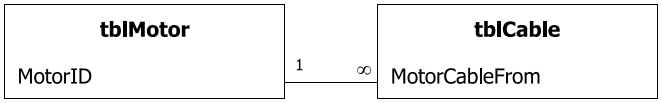
Attributes: Enforced
RelationshipType: One-To-Many

tblIotblIoInterlock



Attributes: Enforced
RelationshipType: One-To-Many

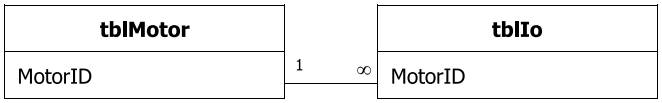
tblMotortblCable



Attributes: Enforced
RelationshipType: One-To-Many

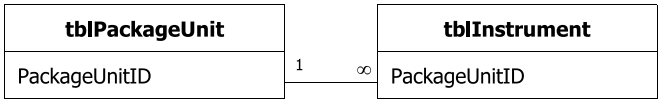
APPENDIX E

tblMotortblIo



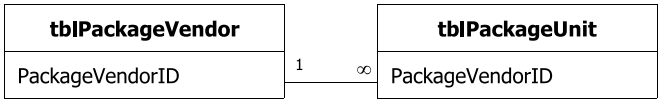
Attributes: Enforced
RelationshipType: One-To-Many

tblPackageUnittblInstrument



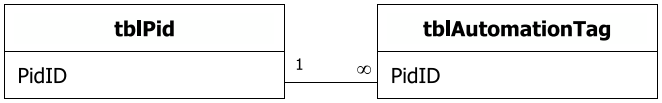
Attributes: Enforced
RelationshipType: One-To-Many

tblPackageVendortblPackageUnit



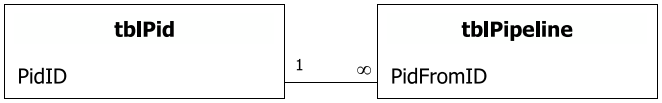
Attributes: Enforced
RelationshipType: One-To-Many

tblPidtblAutomationTag



Attributes: Enforced
RelationshipType: One-To-Many

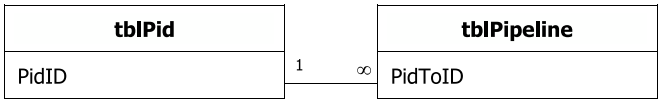
tblPidtblPipeline



Attributes: Enforced
RelationshipType: One-To-Many

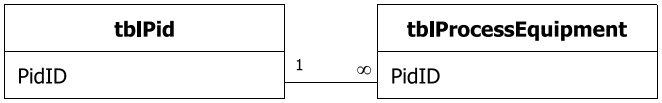
APPENDIX E

tblPidtblPipeline



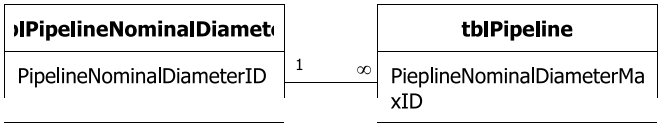
Attributes: Enforced
RelationshipType: One-To-Many

tblPidtblProcessEquipment



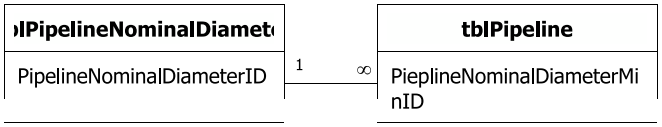
Attributes: Enforced
RelationshipType: One-To-Many

tblPipelineNominalDiametertblPipeline



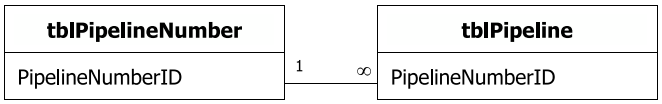
Attributes: Enforced
RelationshipType: One-To-Many

tblPipelineNominalDiametertblPipeline



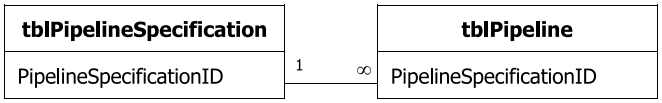
Attributes: Enforced
RelationshipType: One-To-Many

tblPipelineNumbertblPipeline



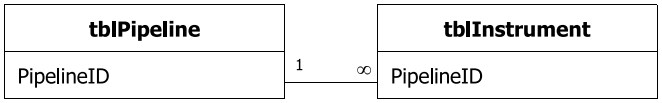
Attributes: Enforced
RelationshipType: One-To-Many

tblPipelineSpecificationtblPipeline



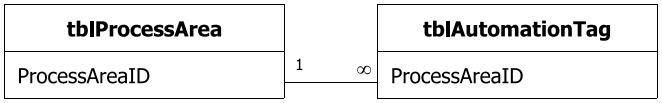
Attributes: Enforced
RelationshipType: One-To-Many

tblPipelinetblInstrument



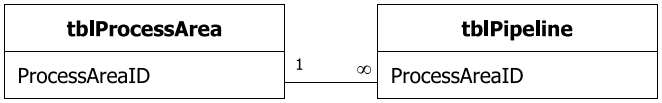
Attributes: Enforced
RelationshipType: One-To-Many

tblProcessAreatblAutomationTag



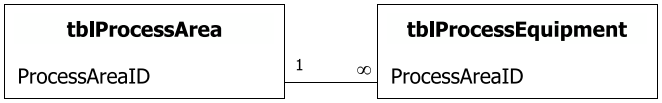
Attributes: Enforced
RelationshipType: One-To-Many

tblProcessAreatblPipeline



Attributes: Enforced
RelationshipType: One-To-Many

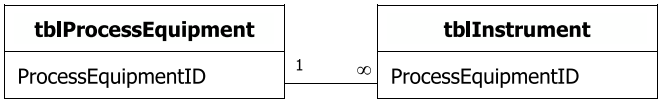
tblProcessAreatblProcessEquipment



Attributes: Enforced
RelationshipType: One-To-Many

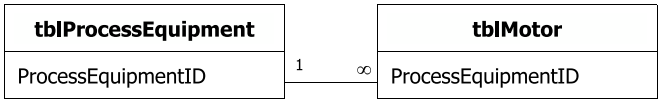
APPENDIX E

tblProcessEquipmenttblInstrument



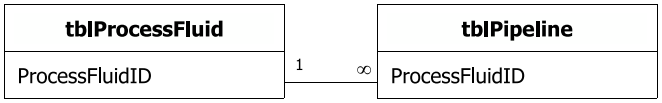
Attributes: Enforced
RelationshipType: One-To-Many

tblProcessEquipmenttblMotor



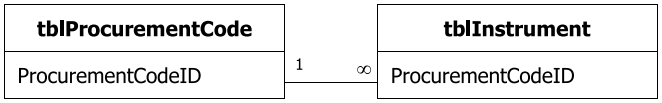
Attributes: Enforced
RelationshipType: One-To-Many

tblProcessFluidtblPipeline



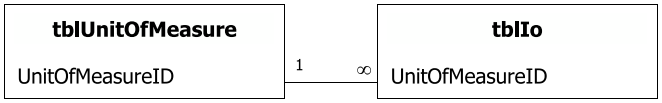
Attributes: Enforced
RelationshipType: One-To-Many

tblProcurementCodetblInstrument



Attributes: Enforced
RelationshipType: One-To-Many

tblUnitOfMeasuretblIo



Attributes: Enforced
RelationshipType: One-To-Many

APPENDIX F

Test Case ID:	TC-1	**Test Case Name:**	Process Equipment
System:	FLUXUS	**Subsystem:**	Process Equipment List
Desined by:	Mika Latva-Kyyny	**Design Date:**	10.3.2016
Executed by:	Mika Latva-Kyyny	**Execution Date:**	4.4.2016
Short description:	Test the Process Equipment List	**Related Use Cases**	UC-1.1, UC-4.1, and UC-4.2

Note: Expected system response is not indicated if it is obvious.

Pre-conditions
Main Menu is open.
Process Equipment List and *Automation Tag List* do not contain any data.
Settings window has default settings.

Step	Input	Expected System Response	Pass/Fail	Comment
1	Open *Equipment List* window		Passed	
2	Select *P&ID* tab		Passed	
3	On a new row, add "PT" to *P&ID Process Area*, "001" to *P&ID Number*, "Title 1" to *Title 1* , "Title 2" to *Title 2* , "Title 3" to *Title 3*, and "Test" to *Internal Notes*	*P&ID Document Number* = "PT-9T-001"	Passed	
4	On a new row, add "CT" to *P&ID Process Area* and "050" to *P&ID Number*	*P&ID Document Number* = "CT-9T-050"	Passed	
5	On a new row, add "CT" to *P&ID Process Area* and "051" to *P&ID Number*	*P&ID Document Number* = "CT-9T-051"	Passed	
6	On a new row, add "PT" to *P&ID Process Area* and "051" to *P&ID Number*	*P&ID Document Number* = "PT-9T-051"	Passed	
7	On a new row, add "PT" to *P&ID Process Area* and "051" to *P&ID Number*	Warning indicating that there is a duplicate value	Passed	
8	On a new row, add "PT" to *P&ID Process Area* and "" to *P&ID Number*	Warning indicating that a value must be entered	Passed	
9	On a new row, add "" to *P&ID Process Area* and "005" to *P&ID Number*	Warning indicating that a value must be entered	Passed	
10	On a new row, add "" to *P&ID Process Area* , "" to *P&ID Number* , "Title 1" to *Title 1*, "Title 2" to *Title 2* , "Title 3" to *Title 3* ,and "Test" to *Internal Notes*	Warning indicating that a value must be entered	Passed	
11	Add "PT" to *P&ID Process Area* and " " to *P&ID Number* on a new row	Warning indicating that a value must be entered	Passed	
12	On a new row, add " " to *P&ID Process Area* and "005" to *P&ID Number*	Warning indicating that a value must be entered	Passed	
13	On a new row, add "BF" to *P&ID Process Area* and "013" to *P&ID Number*	*P&ID Document Number* = "BF-9T-013"	Passed	
14	Change "BF" into "CO" on *P&ID Process Area*	*P&ID Document Number* = "CO-9T-013"	Passed	
15	Mark "CO-9T-013" as *DELETED*	*P&ID Document Number* = "DELETED CO-9T-013"	Passed	
16	Select *Equipment Type* tab		Passed	
17	On a new row, add "C" to *Main Type Code*, "Vessel" to *Main Type Description*, "Column" to *Sub Type Description*		Passed	
18	On a new row, add "C" to *Main Type Code*, "Vessel" to *Main Type Description*, "Vertical Vessel" to *Sub Type Description*		Passed	
19	On a new row, add "D" to *Main Type Code*, "Tank" to *Main Type Description*, "Storage Tank" to *Sub Type Description*		Passed	
20	Change "Storage Tank" into "Sump" on *Sub Type Description*		Passed	
21	On a new row, add "C" to Main Type Code, "Vessel" to *Main Type Description* , "Column" to *Sub Type Description*		Passed	
22	On a new row, add "" to *Main Type Code* , "Vessel" to *Main Type Description* , "Column" to *Sub Type Description*	Warning indicating that a value must be entered	Passed	
23	On a new row, add "C" to *Main Type Code* , "" to *Main Type Description* , "Column" to *Sub Type Description*	Warning indicating that a value must be entered	Passed	
24	On a new row, add "C" to *Main Type Code* , "Vessel" to *Main Type Description* , "Column" to *Sub Type Description*	Warning indicating that a value must be entered	Passed	
25	On a new row, add " " to *Main Type Code* , "Vessel" to *Main Type Description* , "Column" to *Sub Type Description*	Warning indicating that a value must be entered	Passed	
26	On a new row, add "C" to *Main Type Code* , " " to *Main Type Description* , "Column" to *Sub Type Description*	Warning indicating that a value must be entered	Passed	
27	On a new row, add "C" to *Main Type Code* , "Vessel" to *Main Type Description* , " " to *Sub Type Description*	Warning indicating that a value must be entered	Passed	
28	Close *Equipment List* window		Passed	
29	Open *Automation Tag List* window		Passed	
30	Select Process Area tab		Passed	

Step	Input	Expected System Response	Pass/Fail	Comment
31	On a new row, add "PT" to Area Code and "Purified PTA" to Description		Passed	
32	On a new row, add "TF" to Area Code and "" to Description		Passed	
33	On a new row, add "CT" to Area Code and "Crude PTA" to Description		Passed	
34	On a new row, add "Test" to Area Code and "Crude PTA" to Description		Passed	
35	Change "TF" into "BF" on Area Code		Passed	
36	On a new row, add "" to Area Code and "Test" to Description	Warning indicating that a value must be entered	Passed	
37	On a new row, add " " to Area Code and "Test" to Description	Warning indicating that a value must be entered	Passed	
38	On a new row, add "CT" to Area Code and "" to Description	Warning indicating that there is a duplicate value	Passed	
39	Close *Automation Tag List* window		Passed	
40	Open *Equipment List* window		Passed	
41	Select *Equipment List* tab		Passed	
42	On a new row, choose "CT-9T-050" from the drop-down list of *P&ID* , choose "CT" from the drop-down list of *Process Area*, choose "C, Vessel, Vertical Vessel" from the drop-down list of *Type*, add "025" to *Number* , add "A" to *Suffix* , add "Wash Acid Tank" to *Description* , add "New" to *Status* , add "-20" to *Design Temp. Min* , add "200" to *Design Temp. Max* , add "-0,7" to *Design Press. Min* , add 30 to *Design Press. Max* , add "-5,5" to *Oper. Temp. Min* , add "-0,5" to *Oper. Temp. Norm* , add "250" to *Oper. Temp. Max* , add "-0,5" to *Oper. Press. Min* , add "-0,1" to *Oper. Press. Norm* , add "12" to *Oper. Press. Max* , add "MAIN" to *Scope* , add "Test 1" to *Remarks* , and add "Test 2" to *Internal Notes*	*Calc. Tag* = "CT-C-025A"	Passed	
43	On a new row, choose "PT-9T-051" from the drop-down list of *P&ID* , choose "PT" from the drop-down list of *Process Area* , choose "C, Vessel, Vertical Vessel" from the drop-down list of *Type* , and add "5" to *Number*	*Calc. Tag* = "PT-C-5"	Passed	
44	Change "5" into "65" on *Number*	*Calc. Tag* = "PT-C-65"	Passed	
45	Mark equipment "PT-C-65" as *DELETED*	*Calc. Tag* = "DELETED (PT-C-65)"	Passed	
46	Mark equipment "PT-C-65" as *NOT DELETED*	*Calc. Tag* = "PT-C-65"	Passed	
47	Mark equipment "PT-C-65" as *DELETED*	*Calc. Tag* = "DELETED (PT-C-65)"	Passed	
48	On a new row, choose "PT-9T-051" from the drop-down list of *P&ID* , choose "PT" from the drop-down list of *Process Area* , choose "C, Vessel, Vertical Vessel" from the drop-down list of *Type* , and add "" to *Number*	Warning indicating that a value must be entered	Passed	
49	On a new row, choose "PT-9T-051" from the drop-down list of *P&ID* , choose "PT" from the drop-down list of *Process Area* , choose "C, Vessel, Vertical Vessel" from the drop-down list of *Type* , and add " " to *Number*	Warning indicating that a value must be entered	Passed	
50	On a new row, choose "PT-9T-051" from the drop-down list of *P&ID* , choose "PT" from the drop-down list of *Process Area* , choose nothing from the drop-down list of *Type* , and add "44" to *Number*	Warning indicating that a value must be entered	Passed	
51	On a new row, choose "PT-9T-051" from the drop-down list of *P&ID* , choose nothing from the drop-down list of *Process Area* , choose "C, Vessel, Vertical Vessel" from the drop-down list of *Type* , and add "44" to *Number*	Warning indicating that a value must be entered	Passed	
52	On a new row, choose nothing from the drop-down list of *P&ID* , choose "PT" from the drop-down list of *Process Area* , choose "C, Vessel, Vertical Vessel" from the drop-down list of *Type* , and add 44 to *Number*	Warning indicating that a value must be entered	Passed	
53	Close *Equipment List* window		Passed	

Test Case ID:	TC-2	**Test Case Name:**	Pipeline
System:	FLUXUS	**Subsystem:**	Pipeline List
Desined by:	Mika Latva-Kyyny	**Design Date:**	10.3.2016
Executed by:	Mika Latva-Kyyny	**Execution Date:**	4.4.2016
Short description:	Test the Pipeline List	**Related Use Cases**	UC-5.1 and UC-5.2

Note: Expected system response is not indicated if it is obvious.

Pre-conditions
Main Menu is open and TC-1 has been executed.
Pipeline List does not contain any data.
Settings window has default settings.

Step	Input	Expected System Response	Pass/Fail	Comment
1	Open *Line List* window		Passed	
2	Select *Heat Tracing* tab		Passed	
3	On a new row, add "°C" to *Heat Tracing Temperature Unit* and "123" to *Heat Tracing Temperature Value*		Passed	
4	On a new row, add "°C" to *Heat Tracing Temperature Unit* and "" to *Heat Tracing Temperature Value*	Warning indicating that a value must be entered	Passed	
5	On a new row, add "°C" to *Heat Tracing Temperature Unit* and " " to *Heat Tracing Temperature Value*	Warning indicating that a value must be entered	Passed	
6	On a new row, add " " to *Heat Tracing Temperature Unit* and "123" to *Heat Tracing Temperature Value*	Warning indicating that a value must be entered	Passed	
7	On a new row, add "" to *Heat Tracing Temperature Unit* and "123" to *Heat Tracing Temperature Value*	Warning indicating that a value must be entered	Passed	
8	On a new row, add "°C" to *Heat Tracing Temperature Unit* and "123" to *Heat Tracing Temperature Value*	Warning indicating that there is a duplicate value	Passed	
9	On a new row, add "°C" to *Heat Tracing Temperature Unit* and "50.1" to *Heat Tracing Temperature Value*	Warning indicating that the datatype must be integer	Passed	
10	On a new row, add "F" to *Heat Tracing Temperature Unit* and "50,1" to *Heat Tracing Temperature Value*	*Heat Tracing Temperature Value* = "50"	Passed	
11	On a new row, add "F" to *Heat Tracing Temperature Unit* and "60,5" to *Heat Tracing Temperature Value*	*Heat Tracing Temperature Value* = "60"	Passed	
12	On a new row, add "F" to *Heat Tracing Temperature Unit* and "70,51" to *Heat Tracing Temperature Value*	*Heat Tracing Temperature Value* = "71"	Passed	
13	Select *Insulation* tab		Passed	
14	On a new row, add "G2"" to *Insulation Code* , add "Unknown" to *Insulation Material* , and add "2"" to *Insulation Thickness*		Passed	
15	On a new row, add "G2"(test)" to *Insulation Code* , add "Unknown" to *Insulation Material* , and add "2"" to *Insulation Thickness*		Passed	
16	On a new row, add "G4"" to *Insulation Code* , add "Unknown" to *Insulation Material* , and add "4"" to *Insulation Thickness*		Passed	
17	On a new row, add "G6"" to *Insulation Code* , add "" to *Insulation Material* , and add "" to *Insulation Thickness*		Passed	
18	On a new row, add "G2"" to *Insulation Code* , add "" to *Insulation Material* , and add "" to *Insulation Thickness*	Warning indicating that there is a duplicate value	Passed	
19	On a new row, add "" to *Insulation Code* , add "Unknown" to *Insulation Material* , and add "8"" to *Insulation Thickness*	Warning indicating that a value must be entered	Passed	
20	Select *Diameter* tab		Passed	
21	On a new row, add "1/2" to *Nominal Diameter Value and* add """ to *Nominal Diameter Unit*		Passed	
22	On a new row, add "10" to *Nominal Diameter Value and* add """ to *Nominal Diameter Unit*		Passed	
23	On a new row, add "10" to *Nominal Diameter Value and* add "m" to *Nominal Diameter Unit*		Failed	
24	On a new row, add "" to *Nominal Diameter Value and* add "m" to *Nominal Diameter Unit*	Warning indicating that a value must be entered		
25	On a new row, add "2" to *Nominal Diameter Value and* add "" to *Nominal Diameter Unit*	Warning indicating that a value must be entered	Failed	
26	On a new row, add "1/2" to *Nominal Diameter Value and* add """ to *Nominal Diameter Unit*	Warning indicating that there is a duplicate value	Passed	
27	Select *Pipeline Specification* tab		Passed	
28	On a new row, add "C300" to *Specification Code* , add "300" to *Pressure Class* , and add "Carbon Steel" to *Material*		Passed	
29	On a new row, add "C150" to *Specification Code* , add "" to *Pressure Class* , and add "" to *Material*		Passed	
30	On a new row, add "" to *Specification Code* , add "300" to *Pressure Class* , and add "Carbon Steel" to *Material*	Warning indicating that a value must be entered	Passed	
31	On a new row, add "C300" to *Specification Code* , add "" to *Pressure Class* , and add "" to *Material*	Warning indicating that there is a duplicate value	Passed	
32	Select *Pipeline Number* tab		Passed	
33	On a new row, add "1234" to *Pipeline Number*		Passed	
34	On a new row, add "2,1" to *Pipeline Number*	"2"	Passed	

Step	Input	Expected System Response	Pass/Fail	Comment
35	On a new row, add "2,5" to *Pipeline Number*	"2"	Passed	
36	On a new row, add "2,51" to *Pipeline Number*	"3"	Passed	
37	On a new row, add " " to *Pipeline Number*	Warning indicating that a value must be entered	Passed	
38	On a new row, add "1234" to *Pipeline Number*	Warning indicating that there is a duplicate value	Passed	
39	Select *Process Fluid* tab		Passed	
40	On a new row, add "PG" to *Process Fluid Code* and "Process Gas" to *Process Fluid Description*		Passed	
41	On a new row, add "PL" to *Process Fluid Code* and "" to *Process Fluid Description*		Passed	
42	On a new row, add "" to *Process Fluid Code* and "Process Liquid" to *Process Fluid Description*	Warning indicating that a value must be entered	Passed	
43	On a new row, add " " to *Process Fluid Code* and "Process Liquid" to *Process Fluid Description*	Warning indicating that a value must be entered	Passed	
44	On a new row, add "PG" to *Process Fluid Code* and "" to *Process Fluid Description*	Warning indicating that there is a duplicate value	Passed	
45	Select *Pipeline List* tab		Passed	
46	On a new row, select "CT" from the drop-down list of *Process Area* , select "PG" from the drop-down list of *Process Fluid* , select "1234" from the drop-down list of *Pipeline Number* , select "1/2"" from the drop-down list of *Nominal Diameter Min.* , select "10"" from the drop-down list of *Nominal Diameter Max.,* select "C300" from the drop-down list of *Pipeline Specificatio* , select "G2"" from the drop-down list of *Insulation* , select "123" from the drop-down list of *Heat Tracing* , select "CT-9T-050" from the drop-down list of *P&ID (from)* , select "CT-9T-051" from the drop-down list of *P&ID (to)* , add "-0,7" to *Design Pressure Min* , add "21,7" to *Design Pressure Max* , add "-20,5" to *Design Temp. Min* , add "-0,5" to *Design Temp. Max* , add "-0,1" to *Operating Pressure* , add "-0,1" to *Operating Temp* ., add , "Test 1" to *Remarks* , add "Test 2" to *Internal Notes* , add "1" to *Revision* , and choose "New" from the drop-down list of *Pipeline Status*	*Pipeline Tag* = "CTPG-1234"	Passed	Column headings *InsulationID* and *HeatTracingID* to be corrected.
47	On a new row, select "CT" from the drop-down list of *Process Area* , select "PG" from the drop-down list of *Process Fluid* , select "2" from the drop-down list of *Pipeline Number* , and select "1/2"" from the drop-down list of *Nominal Diameter Min.*	*Pipeline Tag* = "CTPG-2"	Passed	
48	Mark pipeline "CTPG-2" as *DELETED*	*Pipeline Tag* = "DELETED (CTPG-2)"	Passed	
49	On a new row, select nothing from the drop-down list of *Process Area* , select "PG" from the drop-down list of *Process Fluid* , select "3" from the drop-down list of *Pipeline Number* , and select "1/2"" from the drop-down list of *Nominal Diameter Min.*	Warning indicating that a value must be entered	Passed	
50	On a new row, select "CT" from the drop-down list of *Process Area* , select nothing from the drop-down list of *Process Fluid* , select "3" from the drop-down list of *Pipeline Number* , and select "1/2"" from the drop-down list of *Nominal Diameter Min.*	Warning indicating that a value must be entered	Passed	
51	On a new row, select "CT" from the drop-down list of *Process Area* , select "PG" from the drop-down list of *Process Fluid* , select nothing from the drop-down list of *Pipeline Number* , and select "1/2"" from the drop-down list of *Nominal Diameter Min.*	Warning indicating that a value must be entered	Passed	
52	On a new row, select "CT" from the drop-down list of *Process Area* , select "PG" from the drop-down list of *Process Fluid* , select "3" from the drop-down list of *Pipeline Number* , and select nothing from the drop-down list of *Nominal Diameter Min.*	Warning indicating that a value must be entered	Passed	
53	Close *Line List* window		Passed	

Test Case ID:	TC-3	**Test Case Name:**	Automation Tag
System:	FLUXUS	**Subsystem:**	Automation Tag List
Desined by:	Mika Latva-Kyyny	**Design Date:**	10.3.2016
Executed by:	Mika Latva-Kyyny	**Execution Date:**	4.4.2016
Short description:	Test the Automation Tag List	**Related Use Cases**	UC-1.1 and UC-1.2

Note: Expected system response is not indicated if it is obvious.

Pre-conditions
Main Menu is open and TC-2 has been executed.
Automation Tag List contains only the data added to *Process Area* tab in TC-1.4.1.
Settings window has default settings.

Step	Input	Expected System Response	Pass/Fail	Comment
1	Open *Automation Tag List* window		Passed	
2	Select *Tag Type* tab		Passed	
3	On a new row, add "TT" to *Tag Type* and "Temperature transmitter" to *Description*		Passed	
4	On a new row, add "PT" to *Tag Type* and "001122334455" to *Description*		Passed	
5	On a new row, add "TI" to *Tag Type* and "Temperature Gauge" to *Description*		Passed	
6	On a new row, add "PI" to *Tag Type* and "Pressure Gauge" to *Description*		Passed	
7	On a new row, add "FT" to *Tag Type* and "Flow Transmitter" to *Description*		Passed	
8	On a new row, add "HV" to *Tag Type* and "Block Valve" to *Description*		Passed	
9	Change "TT" into "TIT" on *Tag Type*		Passed	
10	Change "Temperature transmitter" into "TX" on *Description*		Passed	
11	Remove "001122334455" from *Description*	Warning indicating that a value must be entered	Passed	
12	On a new row, add " " to *Tag Type* and "" to *Description*	Warning indicating that a value must be entered	Passed	
13	On a new row, add " " to *Tag Type* and "Test" to *Description*	Warning indicating that a value must be entered	Passed	
14	On a new row, add "" to *Tag Type* and "Test" to *Description*	Warning indicating that a value must be entered	Passed	
15	On a new row, add "TIT" to *Tag Type* and "TX" to *Description*	Warning indicating that there is a duplicate value	Passed	
16	On a new row, add "FICV" to *Tag Type* and " " to *Description*	Warning indicating that a value must be entered	Passed	
17	On a new row, add "FICV" to *Tag Type* and "" to *Description*	Warning indicating that a value must be entered	Passed	
18	Select *Tag Number* tab		Passed	
19	On a new row, add "12345" to *Tag Number*		Passed	
20	On a new row, add "01234" to *Tag Number*	"1234"	Passed	
21	On a new row, add "10000" to *Tag Number*		Passed	
22	On a new row, add "6" to *Tag Number*		Passed	
23	Change "1234" into "5555" on *Tag Number*		Passed	
24	On a new row, add " " to *Tag Number*	Warning indicating that a value must be entered	Passed	
25	On a new row, add "12345" to *Tag Number*	Warning indicating that there is a duplicate value	Passed	
26	On a new row, add "12345A" to *Tag Number*	Warning indicating that the datatype must be integer	Passed	
27	On a new row, add "X" to *Tag Number*	Warning indicating that the datatype must be integer	Passed	
28	On a new row, add "X6" to *Tag Number*	Warning indicating that the datatype must be integer	Passed	
29	On a new row, add "%" to *Tag Number*	Warning indicating that the datatype must be integer	Passed	
30	On a new row, add "22.5" to *Tag Number*	Warning indicating that the datatype must be integer	Passed	
31	On a new row, add "22,4" to *Tag Number*	"22"	Passed	
32	On a new row, add "22,5" to *Tag Number*	"22"	Passed	
33	On a new row, add "22,51" to *Tag Number*	"23"	Passed	
34	Select *Automation Tag* tab		Passed	
35	On a new row, choose "12345" from the drop-down list of *Tag Number* , choose "TIT" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "Instrument" from the drop-down list of *Tag Category* , add "A" to *Tag Suffix* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID* , add "A6" to *P&ID Coord.* , and add "3" to *P&ID Rev.*	*Calc. Tag* = "TIT-CT-12345A"	Passed	
36	On a new row, choose "10000" from the drop-down list of *Tag Number* , choose "TIT" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "I/O" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	*Calc. Tag* = "TIT-CT-10000"	Passed	
37	On a new row, choose "6" from the drop-down list of *Tag Number* , choose "TI" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "Instrument" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	*Calc. Tag* = "TI-CT-6"	Passed	

Step	Input	Expected System Response	Pass/Fail	Comment
38	On a new row, choose "6" from the drop-down list of *Tag Number* , choose "PI" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "Instrument" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	*Calc. Tag* = "PI-CT-6"	Passed	
39	On a new row, choose "22" from the drop-down list of *Tag Number* , choose "PT" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "Instrument" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	*Calc. Tag* = "PT-CT-22"	Passed	
40	On a new row, choose "23" from the drop-down list of *Tag Number* , choose "FT" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "Instrument" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	*Calc. Tag* = "FT-CT-23"	Passed	
41	On a new row, choose "5555" from the drop-down list of *Tag Number* , choose "HV" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "Instrument" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	*Calc. Tag* = "HV-CT-5555"	Passed	
42	On a new row, choose "22" from the drop-down list of *Tag Number* , choose "HV" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "Instrument" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	*Calc. Tag* = "HV-CT-22"	Passed	
43	On a new row, choose "12345" from the drop-down list of *Tag Number* , choose "HV" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "Instrument" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	*Calc. Tag* = "HV-CT-12345"	Passed	
44	Mark Automation Tag "HV-CT-22" as *DELETED*	*Calc. Tag* = "DELETED (HV-CT-22)"	Passed	
45	On a new row, choose nothing from the drop-down list of *Tag Number* , choose "TIT" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "I/O" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	Warning indicating that a value must be entered	Passed	
46	On a new row, choose "5555" from the drop-down list of *Tag Number* , choose nothing from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "I/O" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	Warning indicating that a value must be entered	Passed	
47	On a new row, choose "5555" from the drop-down list of *Tag Number* , choose "TIT" from the drop-down list of *Tag Type* , choose nothing from the drop-down list of *Process Area,* choose "I/O" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	Warning indicating that a value must be entered	Passed	
48	On a new row, choose "5555" from the drop-down list of *Tag Number* , choose "TIT" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose nothing from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	Warning indicating that a value must be entered	Passed	

Step	Input	Expected System Response	Pass/Fail	Comment
49	On a new row, choose "5555" from the drop-down list of *Tag Number* , choose "TIT" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "I/O" from the drop-down list of *Tag Category* , choose nothing from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	Warning indicating that a value must be entered	Passed	
50	On a new row, choose "5555" from the drop-down list of *Tag Number* , choose "TIT" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "I/O" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose nothing from the drop-down list of *P&ID*	Warning indicating that a value must be entered	Passed	
51	On a new row, choose "10000" from the drop-down list of *Tag Number* , choose "TIT" from the drop-down list of *Tag Type* , choose "CT" from the drop-down list of *Process Area,* choose "I/O" from the drop-down list of *Tag Category* , choose "MAIN" from the drop-down list of *Scope* , choose "CT-9T-051" from the drop-down list of *P&ID*	Warning indicating that there is a duplicate value	Passed	
52	Close *Automation Tag List* window		Passed	

Test Case ID:	TC-4	**Test Case Name:**	Instrument Datasheets
System:	FLUXUS	**Subsystem:**	Instrument Datasheets
Desined by:	Mika Latva-Kyyny	**Design Date:**	10.3.2016
Executed by:	Mika Latva-Kyyny	**Execution Date:**	4.4.2016
Short description:	Test the Instrument Datasheets	**Related Use Cases**	UC-2.1, -2.2, -2.3, and -2.4

Note: Expected system response is not indicated if it is obvious.

Pre-conditions
Main Menu is open and TC-3 has been executed.
Instrument Datasheets do not contain any data.
Settings window has default settings.

Step	Input	Expected System Response	Pass/Fail	Comment
1	Open *Datasheets* window		Passed	
2	Click *Add New Instrument*	Add New Instrument -window opens	Passed	
3	Select "HV-CT-5555" from the drop-down list *Select tag for the new instrument*		Passed	
4	Select "Automatic Valve" from the drop-down list *Select instrument type*		Passed	
5	Click *OK*	Add New Instrument -window closes and the new instrumen appears on the datasheet and instruemnt list	Passed	
6	Click *Add New Instrument*	Add New Instrument -window opens	Passed	
7	Select "FT-CT-23" from the drop-down list *Select tag for the new instrument*		Passed	
8	Select "Flowmeter, magnetic" from the drop-down list *Select instrument type*		Passed	
9	Click *OK*	Add New Instrument -window closes and the new instrumen appears on the datasheet and instruemnt list	Passed	
10	Click *Add New Instrument*	Add New Instrument -window opens	Passed	
11	Select "PI-CT-6" from the drop-down list *Select tag for the new*		Passed	
12	Select "Pressure gauge" from the drop-down list *Select instrument type*		Passed	
13	Click *OK*	Add New Instrument -window closes and the new instrumen appears on the datasheet and instruemnt list	Passed	
14	Click *Add New Instrument*	Add New Instrument -window opens	Passed	
15	Select "PT-CT-22" from the drop-down list *Select tag for the*		Passed	
16	Select "Pressure transmitter" from the drop-down list *Select instrument type*		Passed	
17	Click *OK*	Add New Instrument -window closes and the new instrumen appears on the datasheet and instruemnt list	Passed	
18	Click *Add New Instrument*	Add New Instrument -window opens	Passed	
19	Select "TI-CT-6" from the drop-down list *Select tag for the new instrument*		Passed	
20	Select "Temperature gauge" from the drop-down list *Select instrument type*		Passed	
21	Click *OK*	Add New Instrument -window closes and the new instrumen appears on the datasheet and instruemnt list	Passed	
22	Click *Add New Instrument*	Add New Instrument -window opens	Passed	
23	Select "TIT-CT-12345A" from the drop-down list *Select tag for the new instrument*		Passed	
24	Select "Temperature transmitter" from the drop-down list *Select instrument type*		Passed	
25	Click *OK*	Add New Instrument -window closes and the new instrumen appears on the datasheet and instruemnt list	Passed	
26	Click *Add New Instrument*	Add New Instrument -window opens	Passed	
27	Click *Cancel*	Add New Instrument -window closes	Passed	
28	Click *Add New Instrument*	Add New Instrument -window opens	Passed	
29	Click *OK*	Warning window opens indicating that instrument tag must be selected	Passed	
30	Click *OK*	Warning window closes	Passed	
31	Click *Cancel*	Add New Instrument -window closes	Passed	
32	Click *Add New Instrument*	Add New Instrument -window opens	Passed	
33	Select "TIT-CT-12345A" from the drop-down list *Select tag for*		Passed	
34	Click *OK*	Warning window opens indicating that instrument type must be selected	Passed	
35	Click *OK*	Warning window closes	Passed	
36	Click *Cancel*	Add New Instrument -window closes	Passed	
37	Close *Instrument Datasheets* window		Passed	
38	Open *Instrument List* window		Passed	

Step	Input	Expected System Response	Pass/Fail	Comment
39	Choose *Procurement Codes* tab		Passed	
40	On a new row, add "J0001" to *Procurement Code* and "test" to *Description*		Passed	
41	On a new row, add "J0002" to *Procurement Code* and "" to *Description*		Passed	
42	On a new row, add "" to *Procurement Code* and "test" to *Description*	Warning indicating that a value must be entered	Passed	
43	On a new row, add "J0001" to *Procurement Code* and "" to *Description*	Warning indicating that there is a duplicate value	Passed	
44	Choose *Package Vendors* tab		Passed	
45	On a new row, add "Supplier 1" to *Package Vendor Name*		Passed	
46	On a new row, add "Supplier 2" to *Package Vendor Name*		Passed	
47	On a new row, add " " to *Package Vendor Name*	Warning indicating that a value must be entered	Passed	
48	On a new row, add "Supplier 1" to *Package Vendor Name*	Warning indicating that there is a duplicate value	Passed	
49	Choose *Package Units* tab		Passed	
50	On a new row, choose "Supplier 1" from the drop-down list *Package Vendor* , add "Package 1" to *Package Unit Name* , add "delivered" to *Package Unit Status* , and add "test" to *Package Unit Notes*		Passed	
51	On a new row, choose "Supplier 2" from the drop-down list *Package Vendor* , add "Package 2" to *Package Unit Name* , and add "delivered" to *Package Unit Status*		Passed	
52	On a new row, choose "Supplier 2" from the drop-down list *Package Vendor* , add "Package 3" to *Package Unit Name* , and add "" to *Package Unit Status*	Warning indicating that a value must be entered	Passed	
53	On a new row, choose "Supplier 2" from the drop-down list *Package Vendor* , add "" to *Package Unit Name* , and add "delivered" to *Package Unit Status*	Warning indicating that a value must be entered	Passed	
54	On a new row, choose nothing from the drop-down list *Package Vendor* , add "Package 3" to *Package Unit Name* , and add "" to *Package Unit Status*	Warning indicating that a value must be entered	Passed	
55	On a new row, choose "Supplier 1" from the drop-down list *Package Vendor* , add "Package 1" to *Package Unit Name* , add "not delivered" to *Package Unit Status*	Warning indicating that there is a duplicate value	Passed	
56	Choose *Air Manifolds* tab		Passed	
57	On a new row, add "IA-CT-501" to *Manifold Tag*		Passed	
58	On a new row, add " " to *Manifold Tag*	Warning indicating that a value must be entered	Passed	
59	On a new row, add "IA-CT-501" to *Manifold Tag*	Warning indicating that there is a duplicate value	Passed	
60	Choose *Air Manifold Outputs* tab		Passed	
61	On a new row, choose "IA-CT-501" from the drop-down list of *Manifold* and add "1" to *Output Number*		Passed	
62	On a new row, choose "IA-CT-501" from the drop-down list of *Manifold* and add "2" to *Output Number*		Passed	
63	On a new row, choose "IA-CT-501" from the drop-down list of *Manifold* and add "" to *Output Number*	Warning indicating that a value must be entered	Passed	
64	On a new row, choose "" from the drop-down list of *Manifold* and add "5" to *Output Number*	Warning indicating that a value must be entered	Passed	
65	On a new row, choose "IA-CT-501" from the drop-down list of *Manifold* and add "1" to *Output Number*	Warning indicating that there is a duplicate value	Passed	
66	Choose *Document List* tab		Passed	
67	On a new row, choose "Datasheet" from the drop-down list of *Document Type* , add "3E01-3E-12345" to *Document Number* , and add "Test" to *Document Title*		Passed	
68	On a new row, choose "Hook-up" from the drop-down list of *Document Type* , add "3E01-3E-54321" to *Document Number* , and add "Test" to *Document Title*		Passed	
69	On a new row, choose "Location Plan" from the drop-down list of *Document Type* , add "3E01-3E-5001" to *Document Number* , and add "Test" to *Document Title*		Passed	
70	On a new row, choose "Datasheet" from the drop-down list of *Document Type* , add "3E01-3E-11111" to *Document Number* , and add "" to *Document Title*		Passed	
71	On a new row, choose "Datasheet" from the drop-down list of *Document Type* , add "" to *Document Number* , and add "Test" to *Document Title*	Warning indicating that a value must be entered	Passed	
72	On a new row, choose "" from the drop-down list of *Document Type* , add "3E01-3E-11112" to *Document Number* , and add "Test" to *Document Title*	Warning indicating that a value must be entered	Passed	
73	On a new row, choose "Datasheet" from the drop-down list of *Document Type* , add "3E01-3E-12345" to *Document Number* , and add "" to *Document Title*	Warning indicating that there is a duplicate value	Passed	
74	Choose *Instrument List* tab		Passed	

Step	Input	Expected System Response	Pass/Fail	Comment
75	On a row with tag "HV-CT-5555", choose "ISBL" on *Battery Limit* , add "Test" to *Description* , add "Test" to *Service* , choose "CTPG-1234" from the drop-down list of *Pipeline* , choose "PT-C-5" from the drop-down list of *Process Equipment* , add "New" to *Status* , choose "Supplier 1: Package 1" from the drop-down list of *Package Unit,* add "Field" to *Location* , add "Manufacturer 1" to *Manufacturer* , add "Model 1" to *Model* , choose "J0001" from the drop-down list of *Procurement Code,* add "123456789" to *Manufact. Order No.* , choose "Delivered" from the drop-down list of *Procurement Status* , choose "SIL 1" from the drop-down list of *Safety Integrity Level* , add "non-hazardous zone" to *Hazardous Area Classification* , add "non-Ex" on *Explotion Protection Marking* , choose "LOOP POWERED" from the drop-down list of *Power Supply* , add "M20" to *Cable Entry* , choose "IN-LINE" from the drop-down list of *Category* , choose "P" from the drop-down list of *Instrument Installer* , choose "3E01-3E-12345" from the drop-down list of *Datasheet, choose* "I" from the drop-down list of *Field Cable* , add "Test" to *Remarks* , add "Test" to *Internal Notes* , add "1" to *Revision*		Passed	
76	On a row with tag "FT-CT-23", add " " to *Description*		Passed	
77	Close *Instruemnt List* window		Passed	
78	Open *Datasheets* window		Passed	
79	Choose *Automatic Valve* tab		Passed	
80	Choose *Genereal* subtab		Passed	
81	On the first record "HV-CT-5555", choose the first option of the drop-down list for those fields that have a drop-down list and are empty, and for the rest of the filelds, add "11" to all fields that are empty.		Passed	
82	Choose *Specific* subtab		Passed	
83	On the first record "HV-CT-5555", choose the first option of the drop-down list for those fields that have a drop-down list and are empty, and for the rest of the filelds, add "13" to all fields that are empty. If a field is a checkbox, check it.		Passed	
84	Choose *Process Data* subtab		Passed	
85	On the first record "HV-CT-5555", choose the first option of the drop-down list for those fields that have a drop-down list and are empty, and for the rest of the filelds, add "13" to all fields that are empty.		Passed	
86	Choose *Notes* subtab		Passed	
87	On the first record "HV-CT-5555", add "Test" to all fileds that are empty.		Passed	
88	Choose *Flowmeter Magnetic* tab and repeat steps 80 - 87 also to record "FT-CT-23".		Passed	
89	Choose *Pressure Gauge* tab and repeat steps 80 - 87 also to record "PI-CT-6".		Passed	
90	Choose *Pressure Transmitter* tab and repeat steps 80 - 87 also to record "PT-CT-22".		Passed	
91	Choose *Temperature Gauge* tab and repeat steps 80 - 87 also to record "TI-CT-6".		Passed	
92	Choose *Temperature Transmitter* tab and repeat steps 80 - 87 also to record "TIT-CT-12345A".		Passed	
93	Close *Datasheets* window			

Test Case ID:	TC-5	**Test Case Name:**	I/O List
System:	FLUXUS	**Subsystem:**	I/O List
Desined by:	Mika Latva-Kyyny	**Design Date:**	10.3.2016
Executed by:	Mika Latva-Kyyny	**Execution Date:**	4.4.2016
Short description:	Test the I/O List	**Related Use Cases**	UC-3.1 and -3.2

Note: Expected system response is not indicated if it is obvious.

Pre-conditions
Main Menu is open and TC-4 has been executed.
I/O List does not contain any data.
Settings window has default settings.

Step	Input	Expected System Response	Pass/Fail	Comment
1	Open *I/O List* window		Passed	
2	Click *Add New I/O*	Add New I/O -window opens	Passed	
3	Select "TIT-CT-10000" from the drop-down list *Select tag for the new I/O*		Passed	
4	Click *OK*	Add New I/O -window closes and the new I/O appears on the I/O list	Passed	
5	Click *Add New I/O*	Add New I/O -window opens	Passed	
6	Select nothing from the drop-down list *Select tag for the new I/O*		Passed	
7	Click *OK*	Pop-up message appears indicating that the tag must be selected	Passed	
8	Click *Add New I/O*	Add New I/O -window opens	Passed	
9	Click *Cancel*	Add New I/O -window closes	Passed	
10	Select *Unit of Measure List* tab		Passed	
11	On a new row, add "Temperature" to *Measurement Quantity* and "°C" to *Measurement Unit*		Passed	
12	On a new row, add "Pressure" to *Measurement Quantity* and "bar(g)" to *Measurement Unit*		Passed	
13	On a new row, add "Flow" to *Measurement Quantity* and "kg/h" to *Measurement Unit*		Passed	
14	On a new row, add "Level" to *Measurement Quantity* and "" to *Measurement Unit*	Warning indicating that a value must be entered	Passed	
15	On a new row, add "" to *Measurement Quantity* and "%" to *Measurement Unit*	Warning indicating that a value must be entered	Passed	
16	On a new row, add "Temperature" to *Measurement Quantity* and "°C" to *Measurement Unit*	Warning indicating that there is a duplicate value	Failed	Possible to enter duplicate value
17	Select *Control System List* tab		Passed	
18	On a new row, add "ISBL DCS" to *Control System Name*		Passed	
19	On a new row, add "ISBL PLC" to *Control System Name*		Passed	
20	On a new row, add "OSBL DCS" to *Control System Name*		Passed	
21	On a new row, add "OSBL PLC" to *Control System Name*		Passed	
22	On a new row, add " " to *Control System Name*	Warning indicating that a value must be entered	Passed	
23	On a new row, add "ISBL DCS" to *Control System Name*	Warning indicating that there is a duplicate value	Passed	
24	Select *Interlock List* tab		Passed	
25	On a new row, add "CT-101" to *Interlock Tag* and "Test" to *Interlock Description*		Passed	
26	On a new row, add "CT-102" to *Interlock Tag* and "Test" to *Interlock Description*		Passed	
27	On a new row, add "PT-105" to *Interlock Tag* and "" to *Interlock Description*		Passed	
28	On a new row, add "" to *Interlock Tag* and "Null tag" to *Interlock Description*	Warning indicating that a value must be entered	Passed	
29	On a new row, add "CT-101" to *Interlock Tag* and "Duplicate tag" to *Interlock Description*	Warning indicating that there is a duplicate value	Passed	
30	Select *Interlock List with I/Os* tab		Passed	
31	On a new row, select "TIT-CT-10000" from the drop-down list of *I/O* , select "CT-101" from the drop-down list of *Interlock* , add "Temperature high" to *Interlock Function* , select "SIL 1" from the drop-down-list of *SIL* , and add "150 °C" to *Trip Value*		Passed	
32	On a new row, select "TIT-CT-10000" from the drop-down list of *I/O* , and select "CT-102" from the drop-down list of *Interlock*		Passed	
33	On a new row, select "TIT-CT-10000" from the drop-down list of *I/O* , and select nothing from the drop-down list of *Interlock*	Warning indicating that a value must be entered	Passed	
34	On a new row, select nothing from the drop-down list of *I/O* , and select "PT-105" from the drop-down list of *Interlock*	Warning indicating that a value must be entered	Passed	
35	On a new row, select "TIT-CT-10000" from the drop-down list of *I/O* , and select "CT-101" from the drop-down list of *Interlock*	Warning indicating that there is a duplicate value	Passed	
36	Select *I/O List* tab		Passed	

Step	Input	Expected System Response	Pass/Fail	Comment
37	On the row where *I/O Tag* is "TIT-CT-10000", select "TIT-CT-12345A" from the drop down-list of *Instrument* , select nothing from the drop-down list of *Motor* , select "ISBL DCS" from the drop-down list of *Control System*, select "AI" from the drop-down-list of *I/O Type* , add "4 - 20 mA" to *Signal Type* , select "24 VDC" from the drop-down list of *Signal Voltage* , add "-50" to *Av. Meas. Range (Lower)* , add "800" to Av. Meas. Range (Upper), add "-20" to *Set Meas. Range (Lower)* , add "200" to *Set Meas. Range (Upper),* select "°C" from the drop-down list of *Unit of Measure* , add "130" to *Alarm Trip Value* , add "Field" to *Location* , Add "N/A" to *Controller Type* , add "N/A" to *Remarks* , add "N/A" to *Internal Notes,* and add "1" to *Revision*		Passed	
38	Close *I/O List* window		Passed	

Test Case ID:	TC-6	**Test Case Name:**	Motor List
System:	FLUXUS	**Subsystem:**	Motor List
Desined by:	Mika Latva-Kyyny	**Design Date:**	10.3.2016
Executed by:	Mika Latva-Kyyny	**Execution Date:**	4.4.2016
Short description:	Test the Motor List	**Related Use Cases**	UC-6.1 and -6.2

Note: Expected system response is not indicated if it is obvious.

Pre-conditions
Main Menu is open and TC-5 has been executed.
Motor List does not contain any data.
Settings window has default settings.

Step	Input	Expected System Response	Pass/Fail	Comment
1	Open *Motor List* window		Passed	
2	Select *Feeder Panel List* tab		Passed	
3	On a new row, add "MCC-3E01-1A.3+2" to *Feeder Panel Tag*		Passed	
4	On a new row, add "MCC-3E01-1B.5+1" to *Feeder Panel Tag*		Passed	
5	On a new row, add " " to *Feeder Panel Tag*	Warning indicating that a value must be entered	Passed	
6	On a new row, add "MCC-3E01-1A.3+2" to *Feeder Panel Tag*	Warning indicating that there is a duplicate value	Passed	
7	Select *Motor List* tab		Passed	
8	On a new row, add "PT-GM-77777A" to *Motor Tag* , choose "PT-C-5" from the drop-down list of *Process Equipment* , choose "MCC-3E01-1A.3+2" from the drop-down list of *Feeder Panel*, add "123" to *Nominal Voltage* , add "123" to *Nominal Current* , add "123" to *Nomina Power* , choose "DOL" from the drop-down list of *Starting Method*, choose "H-O-A" from the drop-down list of *Field Control Switch* , add "Ex II 2 G IIC T2" to *Hazardous Area Classification* , add "MCC-3E01-57A" to *Schematic Diagram No.* , choose "ISBL" from the drop-down list of *Battery Limit* , add "WRC" to *Related Process Unit/Area* , and add "1" to *Revision*		Passed	
9	On a new row, add "CT-GM-88888A" to Motor Tag		Passed	
10	On a new row, add " " to Motor Tag	Warning indicating that a value must be entered	Passed	
11	On a new row, add "CT-GM-88888A" to Motor Tag	Warning indicating that there is a duplicate value	Passed	
12	Close *Motor List* window		Passed	

APPENDIX F

Test Case ID:	TC-7	**Test Case Name:**	Cable List
System:	FLUXUS	**Subsystem:**	Cable List
Desined by:	Mika Latva-Kyyny	**Design Date:**	10.3.2016
Executed by:	Mika Latva-Kyyny	**Execution Date:**	4.4.2016
Short description:	Test the Cable List	**Related Use Cases**	UC-7.1 and -7.2

Note: Expected system response is not indicated if it is obvious.

Pre-conditions
Main Menu is open and TC-6 has been executed.
Cable List does not contain any data.
Settings window has default settings.

Step	Input	Expected System Response	Pass/Fail	Comment
1	Open *Cable List* window		Passed	
2	Select *Electrical Enclosure List* tab		Passed	
3	On a new row, select "Cabinet" from the drop-down list of *Category* , and add "DCS-2F00-001" to *Enclosure Tag*		Passed	
4	On a new row, select "Cabinet" from the drop-down list of *Category* , and add "DCS-2F00-002" to *Enclosure Tag*		Passed	
5	On a new row, select "Junction box" from the drop-down list of *Category* , and add "JB-PT1-S501" to *Enclosure Tag*		Passed	
6	On a new row, select "Junction box" from the drop-down list of *Category* , and add "" to *Enclosure Tag*	Warning indicating that a value must be entered	Passed	
7	On a new row, select nothing from the drop-down list of *Category* , and add "JB-PT1-S502" to *Enclosure Tag*	Warning indicating that a value must be entered	Passed	
8	On a new row, select "Junction box" from the drop-down list of *Category* , and add "JB-PT1-S501" to *Enclosure Tag*	Warning indicating that there is a duplicate value	Passed	
9	Select *Electrical Enclosure Terminal Row List* tab		Passed	
10	On a new row, select "DCS-2F00-001" from the drop-down list of *Eelctrical Enclosure Tag,* add "X1" to *Terminal Row Tag* , and check the checkbox in *Intrinsically Safe*		Passed	
11	On a new row, select "DCS-2F00-001" from the drop-down list of *Eelctrical Enclosure Tag,* and add "X2" to *Terminal Row Tag*		Passed	
12	On a new row, select "JB-PT1-S501" from the drop-down list of *Eelctrical Enclosure Tag,* and add "X1" to *Terminal Row Tag*		Passed	
13	On a new row, select nothing from the drop-down list of *Eelctrical Enclosure Tag,* and add "X1" to *Terminal Row Tag*	Warning indicating that a value must be entered	Passed	
14	On a new row, select "JB-PT1-S501" from the drop-down list of *Eelctrical Enclosure Tag,* and add "" to *Terminal Row Tag*	Warning indicating that a value must be entered	Passed	
15	On a new row, select "JB-PT1-S501" from the drop-down list of *Eelctrical Enclosure Tag,* and add "X1" to *Terminal Row Tag*	Warning indicating that there is a duplicate value	Failed	Duplicate value is allowed
16	Select *Cable Type List* tab		Passed	
17	On a new row, add "2X(st)YSWBY-fl" to *Cable Type* and add "24x2x1,3" to *Cable Size*		Passed	
18	On a new row, add "2X(st)YSWBY-fl" to *Cable Type* and add "1x2x1,3" to *Cable Size*		Passed	
19	On a new row, add "2X(st)YSWBY-fl" to *Cable Type* and add "" to *Cable Size*	Warning indicating that a value must be entered	Passed	
20	On a new row, add "" to *Cable Type* and add "24x2x1,3" to *Cable Size*	Warning indicating that a value must be entered	Passed	
21	On a new row, add "2X(st)YSWBY-fl" to *Cable Type* and add "24x2x1,3" to *Cable Size*	Warning indicating that there is a duplicate value	Failed	Duplicate value is allowed
22	Select *Cable List* tab		Passed	
23	On a new row, add "3E01-S501" to *Cable Tag* , choose "2X(st)YSWBY-fl 24x2x1,3" from the drop-down list of *Cable Type* , choose "Instrument Trunk Cable" from the drop-down list of *Category* , choose "JB-PT1-S501: X1" from the drop-down list of *Instrument Trunk Cable From* , choose "DCS-2F00-001: X1" from the drop-down list of Instrument Trunk Cable To, add "Test" to *Description* , add "150" to *Length (m)* , add "Test" to *Remarks* , add "1" to *Revision* and add "Test" to *Internal Notes* .		Passed	
24	On a new row, add "TIT-CT-12345A" to *Cable Tag* , choose "2X(st)YSWBY-fl 1x2x1,3" from the drop-down list of *Cable Type* , choose "Instrument Field Cable" from the drop-down list of *Category* , choose "TIT-CT-12345A" from the drop-down list of *Instrument Field Cable From* , and choose "JB-PT1-S501: X1" from the drop-down list of *Instrument Field Cable To*		Passed	

Step	Input	Expected System Response	Pass/Fail	Comment
25	On a new row, add "PT1-S505" to *Cable Tag* , choose "2X(st)YSWBY-fl 1x2x1,3" from the drop-down list of *Cable Type* , choose "Motor Cable" from the drop-down list of *Category* , choose "PT-GM-77777A" from the drop-down list of *Motor Cable From* , and choose "DCS-2F00-001" from the drop-down list of *Motor Cable To*		Passed	
26	On a new row, add "3E01-S600" to *Cable Tag* , choose "2X(st)YSWBY-fl 24x2x1,3" from the drop-down list of *Cable Type* , choose "General Cable" from the drop-down list of *Category* , add "PB-3E12-01" to *General Cable From* , and add "DCS-3E01-209F" to *General Cable To*		Passed	
27	On a new row, add "3E01-S509" to *Cable Tag* , choose "2X(st)YSWBY-fl 24x2x1,3" from the drop-down list of *Cable Type*		Passed	
28	On a new row, add "3E01-S510" to *Cable Tag* , choose nothing from the drop-down list of *Cable Type*	Warning indicating that a value must be entered	Passed	
29	On a new row, add "" to *Cable Tag* , choose "2X(st)YSWBY-fl 24x2x1,3" from the drop-down list of *Cable Type*	Warning indicating that a value must be entered	Passed	
30	On a new row, add "3E01-S509" to *Cable Tag* , choose "2X(st)YSWBY-fl 24x2x1,3" from the drop-down list of *Cable Type*	Warning indicating that there is a duplicate value	Failed	Duplicate value is allowed
31	On a row where *Cable tag* = "3E01-S509", check checkbox *Deleted*		Passed	
32	Close *Cable List* window		Passed	

Test Case ID:	TC-8	**Test Case Name:**	Reports
System:	FLUXUS	**Subsystem:**	Reports
Desined by:	Mika Latva-Kyyny	**Design Date:**	10.3.2016
Executed by:	Mika Latva-Kyyny	**Execution Date:**	4.4.2016
Short description:	Test the Cable List	**Related Use Cases**	UC-8.1

Note: Expected system response is not indicated if it is obvious.

Pre-conditions
Main Menu is open and TC-7 has been executed.
Settings window has default settings.

Step	Input	Expected System Response	Pass/Fail	Comment
1	Open *Reports* window		Passed	
2	Click *Cancel*	*Reports* window closes	Passed	
3	Open *Reports* window		Passed	
4	Click *I/O List*	I/O List report opens in print-preview mode	Passed	
5	Right-click the I/O list report and select *Print*	Print window opens	Passed	
6	Click *Cancel*	Print window closes	Passed	
7	Click *Temperature Transmitter Datasheets*	Datasheets report open in print-preview mode	Passed	
8	Close datasheets report window		Passed	
9	Close *Reports* window		Passed	